МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютеризованих систем управління

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____Литвиненко О. Є.

«____»_____2020 р.

ДИПЛОМНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ
"МАГІСТР"

Тема: Програмний модуль формування графіку консультацій викладачів університету за допомогою генетичного алгоритму

Виконавець: Корчан І.О.

Керівник: Росінська Г.П.

Нормоконтролер: Тупота Є.В.

Київ 2020

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра комп’ютеризованих систем управління

Спеціальність (спеціалізація) 123 «Комп’ютерна інженерія»

(шифр, найменування)

ЗАТВЕРДЖУЮ

Завідувач кафедри

Литвиненко О.Є.

«_____» _____2020 р.

ЗАВДАННЯ

на виконання дипломної роботи

Корчана Іллі Олеговича

(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема дипломної роботи: Програмний модуль формування графіку консультацій викладачів університету за допомогою генетичного алгоритму

затверджена наказом ректора від « 27 » серпня 2020 р. № 1203

2. Термін виконання роботи: з 5 жовтня 2020 р. по 13 грудня 2020 р.

3. Вихідні дані до роботи: Інформація про мову програмування написання програмного продукту; Інструкція по доступним операційним системам, на яких можна використати програмний продукт

4. Зміст пояснювальної записки:

1) Огляд предметної області;

2) Генетичний алгоритм;

3) Проектування програмного модуля

5. Перелік обов'язкового графічного (ілюстративного) матеріалу:

1) Графічний інтерфейс програмного модулю

2) Тестування програмного модулю

3) Результати роботи програмного модулю для групи СП-324

4) Результати роботи програмного модулю для групи СП-325

5) Результати роботи програмного модулю для групи СП-326

6. Календарний план-графік

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1	Ознайомлення з постановкою завдання дипломної роботи	05.10.2020 – 10.10.2020	
2	Аналіз літератури	10.10.2020 – 14.10.2020	
3	Обґрунтування вибору рішення	14.10.2020 – 28.10.2020	
4	Аналіз існуючих методів вирішення завдання	28.10.2020 – 14.11.2020	
5	Побудова моделей основних процесів	15.11.2020 – 20.11.2020	
6	Розробка програмного забезпечення	21.11.2020 – 02.12.2020	
7	Тестування програмного забезпечення	02.12.2020 – 03.12.2020	
8	Оформлення і друк пояснювальної записки	04.12.2020 – 12.12.2020	
9	Оформлення презентації	13.12.2020 – 13.12.2020	

7. Дата видачі завдання: «05» жовтня 2020 р.

Керівник дипломної роботи (проекту) _____ Росінська Г.П.
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Корчан І.О.
(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Програмний модуль формування графіку консультацій викладачів університету за допомогою генетичного алгоритму»: 91 сторінка, 31 рисунок, 2 таблиці, 25 літературних джерел, 1 додаток.

АЛГОРИТМ, ГЕНЕТИЧНИЙ АЛГОРИТМ, АВТОМАТИЗАЦІЯ, ПРОГРАМНИЙ, ПРОДУКТ, PYTHON, ГРАФІК, ДОСЛІДЖЕННЯ.

Об’єкт дослідження – модель складання графіку консультацій у вузі.

Предмет дослідження – програмний модуль формування графіку консультацій за допомогою генетичного алгоритму.

Мета дипломного проекту – проаналізувати сучасний рівень розвитку моделей складання графіку консультацій в вузах, розглянути існуючі методи вирішення задач обрати та обґрунтувати вибір методу, розробити алгоритм та програмний модуль для складання графіку консультацій, що дозволить скоротити час, який витрачається на складання розкладу та підвищити якість одержуваного розкладу.

Методи дослідження – вивчення принципів роботи генетичного алгоритму, використання об’єктно-орієнтованого програмування при розробці додатку для системи в середовищі програмування *Microsoft Visual Studio Code* за допомогою мови програмування *Python*.

Прогнозні припущення щодо розвитку об’єкта дослідження – розширення можливостей системи шляхом розробки додаткових модулів для масштабування додатку та забезпечення можливості вказувати додаткові критерії для складання графіку, а також додання бази даних для зберігання даних.

У результаті виконання дипломної роботи був проведений аналіз сучасного рівня розвитку моделей складання графіку консультацій, підтверджено актуальність даної теми та прийнято рішення про створення автоматизованої системи. Був складений список вимог до розроблюваного програмного модуля, вивчена предметна область, обрано і проаналізовано алгоритм, реалізований в системі. Було

сформульовано задачу оптимізації графіку в рамках генетичних алгоритмів. На основі цієї задачі було реалізовано програмний продукт на мові програмування Python. Результатом роботи є створена автоматизована система, що застосовується для генерації графіку консультацій, використання якого значно знижує часові витрати.

Матеріали дипломної роботи рекомендується використовувати у навчальних цілях.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

ГА – *генетичний алгоритм* – процедура пошуку, що відноситься до еволюційних алгоритмів, яка використовується для вирішення задач оптимізації і моделювання, заснована на механізмах природного відбору і спадкоємства. У них використовується еволюційний принцип виживання найбільш пристосованих особин (відбір, комбінування і варіації шуканих параметрів з використанням механізмів, що нагадують біологічну еволюцію).

Ген – *властивість, знак, детектор* – атомарний елемент генотипу, зокрема, хромосоми.

Хромосома – структурний елемент клітинного ядра біологічних організмів, який є носієм генів у клітинному ядрі особини.

Популяція – це кінцева множина особин, що схрещуються між собою.

Схрещування – генетичний оператор, що відповідає за передачу ознак від батьків до потомків.

Селекція – полягає у виборі (по розрахованим значенням функції пристосованості) тих хромосом, які братимуть участь у створенні нащадків для наступної популяції, тобто для чергового покоління. Такий вибір здійснюється згідно з принципом природного добору, за яким найбільші шанси на участь у створенні нових особин мають хромосоми з найбільшими значеннями функції пристосованості.

Мутація – генетичний оператор, що відповідає за зміну генів особини в допустимих значеннях для покращення його цільової функції.

Фітнес-функція – функція оцінки генетичного алгоритму, що визначає міру пристосованості отриманого рішення.

Жорсткі обмеження – обмеження, які повинні неодмінно задовольнятися, такі які

фізично не можуть бути порушені.

М'які обмеження – обмеження, які можна порушувати, але це порушення повинно бути зведене до мінімуму. Їхнє виконання не є таким же обов'язковим, як жорстких.

МОГА – Мультиоб'єктний генетичний алгоритм

ВСТУП

Складання графіку занять є одним із важливих завдань, що вирішуються при плануванні навчального процесу. В першу чергу, це пов'язано з тим, що без графіку занять неможливе функціонування будь-якого навчального закладу. Іншими словами, графік навчальних занять має бути складено своєчасно. Крім того, розклад повинен бути "якісно" складеним, тобто відповідати ряду вимог і критеріїв. В якості таких критеріїв можуть виступати такі, що відображають економічну ефективність використання наявних ресурсів освітньої системи, комфортність навчання студентів, обмеження за часом навчання і т.д.

У рамках університету графік консультацій також є критичним. Для викладача може стати проблемою питання поєднання розкладу навчальних занять із графіком додаткових консультацій. До того ж, необхідно враховувати навантаження на викладача та студентів. Тому, у даній роботі створено систему, яка допоможе проводити консультації у час, вільний від занять за автоматично сформованим графіком.

Розвиток досліджень, спрямованих на вирішення завдання складання розкладу, можна розбити на два етапи. Перший етап має початок в 80-і роки і закінчується в середині 90-х. У цей період масштабно застосовуються класичні методи вирішення завдань цілочисельного програмування: метод повного перебору, метод розмальовки графа, метод гілок та меж.

Методи, що використовуються у цих розробках, мають високу ступінь формалізації як самого завдання, так і алгоритмів, що використовуються. Застосування класичних методів в освітніх системах навчання стає малоефективним з огляду на велику розмірність задачі і значних часових витрат. Це призвело до появи методів, що одержали назву інтелектуальних, і це був початок другого етапу. В основі цих методів лежить використання різних евристик і евристичних алгоритмів. Рішення задачі складання розкладу з допомогою евристик не гарантує знаходження глобального оптимуму.

Існує ряд робіт, які використовують для автоматизації складання розкладу математичний апарат нечіткої логіки. Нечітка логіка дозволяє помітно спростити формалізацію вимог, але часто призводить до побудови розкладу, що має не найкращі характеристики в результаті переходу від "жорстких" вимог до більш "м'яких".

В даний час для рішення задачі складання розкладу застосовується ще один новий підхід – нейронні мережі. Найважливішим недоліком застосування цього підходу є складність вибору початкового стану нейронної мережі.

В останні роки особливого поширення набули дослідження методів еволюційного пошуку. Застосування методів еволюційного пошуку призводить до отримання хороших результатів, однак має місце висока обчислювальна трудомісткість і відносна неефективність на заключних етапах еволюції. Одним з таких методів є генетичний алгоритм, який добре підходить для складання різноманітних розкладів.

В даній роботі проводиться дослідження, спрямоване на розв’язання завдання складання графіку консультацій викладачів з використанням генетичного алгоритму.

Генетичні алгоритми – це дуже популярні в даний час способи розв’язання задач оптимізації. У їхній основі лежить використання еволюційних принципів для пошуку оптимального розв’язання. Уже сама ідея виглядає досить інтригуючою і цікавою, щоб перетворити її в життя, а численні позитивні результати тільки розпалюють інтерес з боку дослідників.

Метою дипломної роботи є автоматизація, спрощення та пришвидшення роботи кафедри (комп’ютеризованих систем управління) та розробка автоматизованої системи складання графіку консультацій, що дозволяє скоротити час, що витрачається на складання розкладу, і підвищити якість одержуваного розкладу.

Для досягнення цілі необхідно вирішити такі завдання:

– Проаналізувати інформаційні технології та програмні засоби, що використовуються в сфері освіти;

– Дослідити особливості використання генетичного алгоритму при вирішенні задач складання графіків;
– Розробити модель складання графіку консультацій у вузі;
– Розробити програмний модуль на основі генетичного алгоритму, який здійснює пошук оптимального варіанту.

Об’єкт дослідження – модель складання графіку консультацій у вузі.

Предмет дослідження – програмний модуль формування графіку консультацій за допомогою генетичного алгоритму.

При вирішенні поставлених завдань використовувалися методи математичного моделювання, методи і алгоритми еволюційного моделювання, методи структурного моделювання, методи об'єктно-орієнтованого програмування і об'єктно-орієнтованого проектування.

У роботі проведена формалізація організаційних компонентів навчального процесу, таких як: складання графіку і побудова структурних моделей системи. Крім того, досліджено історію появи, основні переваги і недоліки генетичних алгоритмів, що використовуються для вирішення різноманітних задач.

Практична цінність роботи полягає в можливості використання розробленого програмного забезпечення для прийняття рішень при побудові графіку консультацій викладачів і його застосування для отримання розкладу у вузі.

РОЗДІЛ 1
ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 . Інформаційні технології в сфері освіти

Інформатизація сфери освіти – один із пріоритетних напрямків процесу розвитку суспільства. Підвищення ефективності навчального процесу неможливо досягти без використання автоматизованих робочих місць для вирішення завдань, що виникають у ВНЗ. До таких завдань відносяться облік персоналу, матеріально-технічних ресурсів, управління документацією, розподіл викладацького навантаження і т.д. Особливо слід відзначити важливість проблеми створення інформаційного забезпечення для організаційних процесів навчальних закладів, від якого певною мірою залежить ефективність використання науково-педагогічного потенціалу. У цей напрям включені завдання призначення і розподілу навчального навантаження студентів і викладачів – складання розкладу занять. Ця складова навчального процесу регламентує трудовий ритм і впливає на творчу віддачу викладачів і студентів, тому її можна розглядати як фактор оптимізації використання обмежених трудових ресурсів. Технологію ж розробки розкладу слід сприймати не тільки як трудомісткий технічний процес або об'єкт автоматизації з використанням ЕОМ, але і як процес оптимального управління. Таким чином, завдання отримання оптимальних розкладів занять мають очевидний соціальний і економічний ефект.

1.2. Актуальність даної теми

Бажання контролювати ті чи інші аспекти свого життя є одним з факторів, які безпосередньо впливають на задоволення базових потреб людини. Звідси йде постійне прагнення до пізнання, систематизації та класифікації.

Щодня людина стикається з необхідністю планування свого дня. Прикладом може служити як виконання базових повсякденних справ (прийоми їжі, сон,

дотримання гігієни), так і якихось більш комплексних. Плануючи черговий день, ми складаємо якийсь розклад, де враховуємо тривалість тієї чи іншої дії, важливість і разом з тим – порядок їх виконання.

Розклади супроводжують нас всюди: розклад кіносеансів, поїздів, авіарейсів, занять в школі або на курсах, телепрограма, години прийому лікарів і так далі. При складанні списку справ ми орієнтуємося на необхідність кожного з його елементів, зіставляємо тимчасові рамки, ґрунтуючись на обмеження розкладів роботи або надання послуг іншими людьми або компаніями.

Наприклад, якщо останній поїзд додому їде з вашої станції в 0:10, то необхідно скласти свій план і маршрут так, щоб обов'язково встигнути на нього (бажано завчасно, враховуючи можливість появи непередбачених обставин), а ось прочитати кілька глав улюбленої книги можна вже сидячи в цьому поїзді, тому що більш важливою метою є приїзд додому.

Якщо з плануванням свого дня людина справляється майже інтуїтивно, то що робити зі складним, що включає в себе безліч елементів і обмежень розкладом тих же самих авіарейсів або занять в школі? Як правильно розставити елементи, заздалегідь усуваючи можливі майбутні несуттєві або катастрофічні проблеми? Як при цьому зробити рішення максимально вигідним і таким, що влаштовує всіх тих, хто задіяний в даному розкладі?

У таких складних проблемах, як планування, немає одного найкращого способу дійти остаточної відповіді, тому доводиться вдаватися до її пошуку, намагаючись знайти «хорошу» відповідь. Для розв’язання задачі планування найчастіше використовують евристичні алгоритми для пошуку оптимального рішення. Евристичні методи пошуку стають все менш придатними, оскільки вхідні дані стають більш складними та різноманітними. Цей тип проблем відомий в інформатиці як *NP*-повні задачі. Це означає, що не існує відомих алгоритмів пошуку оптимального рішення за поліноміальний час.

Генетичні алгоритми добре підходять для вирішення завдань виробничого планування, оскільки на відміну від евристичних методів генетичні алгоритми діють на сукупність рішень, а не на одне рішення. У виробничому плануванні ця

сукупність рішень складається з багатьох відповідей, які можуть мати різні, іноді суперечливі цілі. Наприклад, в одному рішенні ми можемо оптимізувати виробничий процес, який буде завершено за мінімальний проміжок часу. В іншому рішенні ми можемо оптимізувати для мінімальної кількості дефектів. Підвищуючи швидкість, з якою ми виробляємо, ми можемо зіткнутися зі збільшенням дефектів нашого кінцевого продукту.

Збільшуючи кількість цілей, яких ми намагаємось досягти, ми також збільшуємо кількість обмежень на проблему і аналогічно збільшуємо складність. Генетичні алгоритми ідеально підходять для таких типів задач, де простір пошуку великий, а кількість можливих рішень невелика.

Говорячи про розклад навчальних занять, варто пам'ятати, що правильно складений розклад сприяє поліпшенню сприйняття матеріалів студентами, а значить, сильно впливає на якість отримуваних знань і досвіду. Процес складання розкладів подібного роду має безліч "підводних каменів", обмежень, специфічних умов, які необхідно враховувати для того чи іншого закладу.

Довгий час складанням розкладів займалися вручну, однак, це трудомісткий і важкий процес, що вимагає чималу частку досвіду. На жаль, навіть найдосвідченіший диспетчер не завжди може скласти розклад, який задовольняє всіх його учасників. Прискорити процес і поліпшити якість одержуваного на виході розкладу допоміг би чітко вибудуваний, протестований алгоритм, а якщо цей алгоритм реалізувати ще й програмно, то затрачуваний час значно скоротиться.

Починаючи з другої половини XX століття було проведено багато досліджень алгоритмів, спрямованих на оптимізацію графіків виробництва для середовищ типу "цех роботи" та "лінія потоку". На даний момент питання залишається актуальним, оскільки ідеального рішення ще не знайдено. Бажання шукати це рішення зумовлене великою залежністю продуктивності та прибутковості виробника від оптимізації графіка.

З кінця 1980-х років зростає інтерес до генетичних алгоритмів - алгоритмів оптимізації, заснованих на принципах природної еволюції. Оскільки вони засновані на еволюційному навчанні, вони потрапляють під рубрику штучного інтелекту.

Вони широко використовуються для оптимізації параметрів, класифікації та навчання у широкому діапазоні застосувань.

Придатність для великих просторів пошуку є корисною перевагою при роботі з графіками, що збільшуються, оскільки простір рішень буде швидко зростати, особливо коли це ускладнюється такими функціями, як альтернативні одиниці. Важливо, щоб ці великі пошукові простори пройшли якомога швидше, щоб забезпечити практичну та корисну реалізацію автоматизованої оптимізації розкладу.

Якщо оптимізацію зробити швидко, тоді керівники навчального закладу можуть випробувати сценарії «що-якщо» та детальний аналіз чутливості, а також зможуть якомога швидше реагувати на «кризи». Тому традиційні підходи до оптимізації розкладу, такі як математичне програмування та розгалуження та обмеження, дуже повільні з боку обчислення в такому масивному просторі пошуку.

Скорочення витрат часу може бути наслідком як від повної автоматизації процесу складання розкладу, бізнес-процес якого зображений на рис. 1.1, так і від автоматизації окремих етапів даного процесу.

Рис. 1.1. Бізнес-процес складання розкладу ВНЗ

В даний час перед ВНЗами стоїть наступний вибір:

– продовжувати складати розклад вручну;

– придбати відповідне ПЗ у сторонньої організації;

– реалізувати систему автоматизації процесу складання розкладу особисто.

Ручне складання розкладу має свої недоліки через людський фактор. Мозок людини не здатний з тією ж легкістю як комп'ютер обробляти велику кількість інформації. Подібна робота довіряється тільки досвідченим диспетчерам, але і вони не завжди здатні скласти оптимальний варіант розкладу.

Укладення договору зі сторонніми організаціями, що надають необхідне програмне забезпечення хоч і є кращим варіантом з точки зору результату, але також має свої недоліки. Перш за все, основним недоліком є фінансова сторона питання, так як потрібно вкладення коштів не тільки на придбання системи, але і на її впровадження та підтримку.

Ще одним важливим недоліком даного підходу є той факт, що більшість систем не мають можливості врахування індивідуальних особливостей кожного закладу, що, безсумнівно, дуже сильно впливає на якість одержуваного розкладу.

Саме тому все більше і більше ВНЗ з часом відмовляються від ручного способу і від придбання системи у стороннього виробника і все частіше схиляються до варіанту власної реалізації.

Позитивною стороною в такій ситуації є те, що реалізована система буде максимально "заточена" під конкретний заклад, що збільшує ймовірність створення ще більш оптимального розкладу. Також витрати на дану систему будуть в рази менші, ніж від покупки стороннього ПО. Важливою частиною даного підходу є необхідність в ретельному вивченні і аналізі предметної області, а також потрібна наявність необхідних для реалізації, впровадження та підтримки системи розробників.

Вибір одного з трьох підходів є індивідуальним і залежить від конкретного ВНЗ. Його голова має прийняти рішення про необхідність і економічну вигоду появи подібної системи в своєму закладі.

Складемо список можливих вимог до майбутньої системи:

– Система повинна бути кросплатформною (дана умова є обов'язковою, щоб уникнути проблем з впровадженням і підтримкою системи на різних операційних системах);
– Повинна бути можливість визначати й змінювати необхідні для генерації розкладу даних;
– Повинна бути можливість налаштування важливості і обов'язковості критеріїв генерації;
– Система повинна дозволяти вносити зміни в отриманий розклад (можливість редагування збільшує оптимальність розкладу);
– Часові показники, що витрачаються системою на генерацію розкладу, повинні бути менші, ніж при ручному методі генерації або використанні інших програм, а якість результату – має їм не поступатися.

Дотримання всіх перелічених вимог при розробці дозволить створити універсальну систему, застосування якої стане можливим в будь-якому ВНЗ, що дозволить складати максимально близький до оптимального варіанту розклад.

Правильно організований навчальний процес (коректно складений розклад в тому числі) є важливою складовою освітньої галузі, тому починаючи з середини минулого століття до наших днів питання про створення і автоматизації процесу складання розкладів все ще є актуальним з багатьох причин (наприклад, постійно розвивається і змінюється система освіти, технологічні прогреси та інше).

1.3. Історія автоматизації складання розкладу

Питання про способи створення оптимального, швидкого і якісного алгоритму складання розкладу є актуальним вже довгі роки, однак, суміжні з ним питання планування з'явилися ще раніше.

Як вказано в джерелі [1], ще в 1784 році у Великобританії була здійснена успішна спроба складання розкладу відправлення маршрутних возів, що є одним з перших згадок про розклад в історії. Пізніше в 1840 році з'явився розклад паровозів,

що включає в себе не тільки час відправлення, але також і прибуття. Через 7 років в результаті договору між усіма паровозними компаніями Великобританії визначилося загальний час для країни за Гринвічем.

Однак різке зростання темпу життя людей, розвиток науки і техніки робили процеси складання розкладу і планування занадто важкими зважаючи на велику кількість залежностей і учасників. Тому вже на початку 20-го століття стало ясно, що необхідно терміново вирішувати наростаючі проблеми в даній сфері.

Приблизно в 1903 році відомий діяч в сфері менеджменту Генрі Лоуренс Гант в одній зі своїх робіт [2] представив метод графічного впорядкування робіт (схему календарного плану), який, як помічено в джерелі [3], він спочатку використовував особисто під час звітів перед начальством про виконані роботи. Даний підхід став настільки успішним, що Гант вирішив продовжити його розвиток, чому присвятив залишок свого життя (до 1919 року). Таким чином Гант вніс своє ім'я в історію, як творець знаменитих діаграм Ганта, що є стандартом донині, іменованих на честь його творця.

Саме з моменту створення діаграм Ганта бере свій початок теорія розкладів. Вона є одним з розділів так званого дослідження операцій (ДО). У джерелі [4] дається таке визначення ДО: дисципліна, що займається розробкою і застосуванням методів знаходження оптимальних рішень на основі математичного моделювання, статистичного моделювання та різних евристичних підходів в різних областях людської діяльності. З визначення ДО слідує, що воно спрямоване на пошук і розробку кращих рішень, що є важливою частиною ТР.

Відповідно до джерела [5], теорія розкладів – це розділ дискретної математики, що займається проблемами впорядкування. Сам же термін "Теорія розкладів", як сукупність всіх теоретичних і практичних завдань, з'явився пізніше в 1956 році в роботі [6] американського математика Річарда Беллмана, що став відомим, завдяки своїм видатним заслугам в сфері математики і обчислювальної техніки. В одній з його книг [7] дається таке визначення ТР: "Теорія розкладів – це розділ дослідження операцій, в якому будуються і аналізуються математичні моделі

календарного планування (тобто упорядкування в часі) різних цілеспрямованих дій з урахуванням цільової функції і різних обмежень".

Ще одним відомим прикладом з початку історії ТР є заслуги американського промисловця Генрі Форда. У статті з його біографією [8] сказано, що в 1908 році він став першим, хто почав використовувати промисловий конвеєр для поточного виробництва. Незважаючи на існування конвеєрного методу задовго до появи Форда, він першим використовував даний підхід для технічно складної продукції (автомобіля), що дозволило скоротити час випуску в 1,5 рази.

Далі, починаючи з 50-х років 20-го століття, багато вчених внесли свій вклад в розвиток теорії ТР. Їх головною метою була класифікація і визначення рівня складності завдань ТР, а також визначення правил знаходження оптимального розкладу для простих моделей.

Одним з проривів став алгоритм, заснований на завданні Беллмана, також відомий, як алгоритм Беллмана-Джонсона, який, відповідно до джерела [9], був створений для пошуку оптимальної послідовності запуску деталей в обробку для двох верстатів при послідовній обробці на потокової лінії. Також це завдання розглядали для трьох і більше верстатів, в результаті чого був зроблений висновок про те, що з ростом числа верстатів експоненціально підвищується і складність вирішення таких завдань, що призводить до висновку про необхідність пошуку інших шляхів вирішення поставленої проблеми.

Аж до 70-х років минулого століття багато дослідників займалися питанням складання ефективного та точного алгоритму з обмеженим часом роботи в залежності від довжини вхідної інформації. Великим успіхом було вирішення питання про знаходження найкоротшого шляху між двома вершинами довільного графа, побудова кістякового дерева мінімальної ваги і багато інших завдань.

Великий внесок в області теоретичних і практичних завдань складання розкладу внесли Конвей Р.В., Максвелл В.Л. і Міллер Л.В. в 1967 році, випустивши монографію "Теорія розкладів" [11], а вже в 1975 році радянські вчені Танаєв В.С. і Шкурба В.В. випустили в світ книгу "Введення в теорію розкладу" [12]. З цього моменту можна вважати ТР повністю сформованим розділом.

У 1979 році в одній зі статей, авторами якої стали Лоуле, Грехем, Ленстра і Рінной Кан [13], була вперше представлена класифікація задач ТР, а також модифікована система позначень. Незважаючи на те, що стаття була випущена багато років тому, в перебігу яких вона зазнала безліч змін, дана система є актуальною і до цього дня.

Говорячи про складність алгоритмів розв'язання задач ТР, варто відзначити статтю, випущену ще в 1965 році Едмондсоном [14], в якій він не тільки представляв алгоритм по знаходженню досконалого паросполучення в довільному графі, а й вказував на високу важливість відмінності завдань з точними поліноміальними алгоритмами від тих, для яких відомі лише експоненціальні алгоритми, підставою яких є методи неявного перебору. Важливим припущенням було неможливість побудови поліноміальних алгоритмів для більшості важковирішуваних завдань.

Важливим фактором, що вплинув на визначення складності завдань ТР, є робота Кука [15], випущена в 1971 році, в якій він визначає клас NP, вводить поняття NP-повноти і вказує на те, що завдання здійсненності також є NP-повною. Відповідно до джерела [16], класом NP в теорії алгоритмів прийнято вважати безліч проблем здійсненності, для перевірки рішення яких можна скористатися машиною Тьюринга за час, який не перевищує полінома від розміру вхідних даних, при умові наявності деякої додаткової інформації. А NP-повна задача, що належить класу NP, є завданням, що має відповідь 0 або 1, а знаходження алгоритму її рішення за поліноміальний час означає наявність можливості настільки ж швидкого вирішення будь-якої іншої задачі класу NP.

В результаті подальших досліджень Карпа і Левіна NP-повнота була виявлена і в інших комбінаторних задачах. Багато дослідників стали шукати докази NP-повноти нових і старих завдань в комбінаторній оптимізації. У підсумку в кінці 1970-х була опублікована монографія Гері і Джонсона [17], що включає в себе список, що містить більше трьохсот NP-повних задач, частина з яких (близько двадцяти) ставилася до ТР.

Пошуки доказів *NP*-складності завдань ТР велися аж до виходу статті Пападімітріоу і Янакакіса в 1991 році [18], в якій був нарешті представлений перший теоретичний підхід для вивчення обчислювальної складності в побудові наближених алгоритмів задач оптимізації, а також визначені класи цієї складності. Дані висновки збільшили інтерес дослідників до питання ТР в цілому, і до цього дня вчені всього світу знаходяться в пошуку відповідей.

1.4. Аналіз існуючих програм для вирішення задачі складання розкладу

Сьогоднішня ситуація на ринку така, що дуже мало програмних продуктів відповідає необхідній функціональності, яка потрібна для складання розкладу у ВНЗ. Такий стан речей легко пояснюється тим, що шкільна освіта на сьогоднішній день більш "стандартизована" (в сенсі організації навчального процесу), ніж вузівська. Така стандартизація веде до великого обсягу потенційного ринку продажів копій програмного забезпечення, що призводить до окупності продукту.

У разі ВНЗ попит на системи складання розкладів навіть більше, ніж для шкіл, але має місце велика специфіка організації навчального процесу в кожному окремо взятому вузі. Створити уніфіковане програмне забезпечення не представляється можливим, а вартість створення спеціалізованого продукту у сторонніх розробників виявляється невиправдано велика. Таким чином, поки що жоден програмний продукт не дозволяє досить просто управляти навчальним процесом навіть в період сесії.

Розглянемо можливості найбільш популярних на ринку програмних продуктів, що реалізують рішення задачі складання розкладу.

1.4.1. *aSc Timetables (IBN)*

Програма розроблена для всіх типів початкових і середніх навчальних закладів. Серед основних характеристик розробники виділяють: оптимальне використання кабінетів та інших шкільних приміщень, дотримання всіх існуючих

психологічних і гігієнічних вимог, усунення можливих помилок і суб'єктивних факторів при складанні розкладів в школі, врахування побажань вчителів і мала кількість вікон, комфортний інтерфейс, що дозволяє якомога ефективніше вводити дані.

Крім автоматичного складання розкладу по класам, програма складає окремі розклади для кабінетів і вчителів. Існує можливість архівації та зберігання кількох копій існуючих розкладів, можливість заміни вчителів на період хвороби або відсутності.

Побудова ж розкладу у ВНЗ цією програмою не передбачено.

1.4.2. *AVTOR-2*

Програма призначена для складання розкладів занять і супроводу їх протягом всього навчального року. *AVTOR-2* – універсальна система. Є кілька версій програми, що розраховані на будь-які навчальні заклади. *AVTOR-2* має приємний дизайн і дружній інтерфейс. Програма досить проста в освоєнні. Є докладний посібник, в якому описані всі можливості і способи роботи з програмою. Програма працює на будь-яких *IBM*-сумісних комп'ютерах. Операційна система: *MS DOS*, або *Windows*. Час роботи залежить від розміру навчального закладу і потужності комп'ютера.

Недоліками програми є нестандартизований інтерфейс та необхідність ручної корекції готового розкладу.

1.4.3. Методист

Основні можливості програми: розподіл і контроль навчального навантаження, облік методичних рекомендацій і особистих побажань викладачів ("вікна", методичні дні, небажані уроки і дні тижня), складання розкладу для будь-якого типу навчального закладу – тижневе або семестрове (цикл складає від 1 до 23 тижнів), облік об'єднання груп (класів) в потоки і розбиття класів на підгрупи,

закріплення спеціальних аудиторій (комп'ютерні класи, лінгафонні кабінети, спортзал і т.п.), облік часу переходів між корпусами; вказівка причин "невдалого призначення" занять (зайнята бажана аудиторія, викладач призначений в небажаний для нього день тижня) з можливістю їх "ручного" виправлення (виходить, в цьому випадку програма не може автоматично вирішувати такі проблеми), можливість багаторазового "поліпшення" розкладу (автоматично або "вручну"), можливість вказівки рівня значущості враховується при складанні розкладу чинників, можливість введення пріоритетів викладачів – ступеня обліку їх індивідуальних побажань.

Обмеження функціональності "Методиста": багатозмінні розклади обмежені максимальною кількістю уроків в день – 7, заняття завжди починаються з першого уроку / пари (при необхідності можливе призначення на першу пару "вільного заняття"), не враховується час змін (наприклад для перевірки можливості переходу між корпусами), не враховується "рівень складності" занять для їх раціонального розподілу по тижню, тривалість занять постійна (неможливе складання розкладу для 30 хв. уроку в молодших і 45 хв. – в старших класах).

1.5. Існуючі способи представлення розкладів

Ґрунтуючись на даних джерела [20], на сьогоднішній день існує 3 основних способи подання розкладів:

– графічний (прикладом можуть служити діаграми Ганта (рис. 1.2));

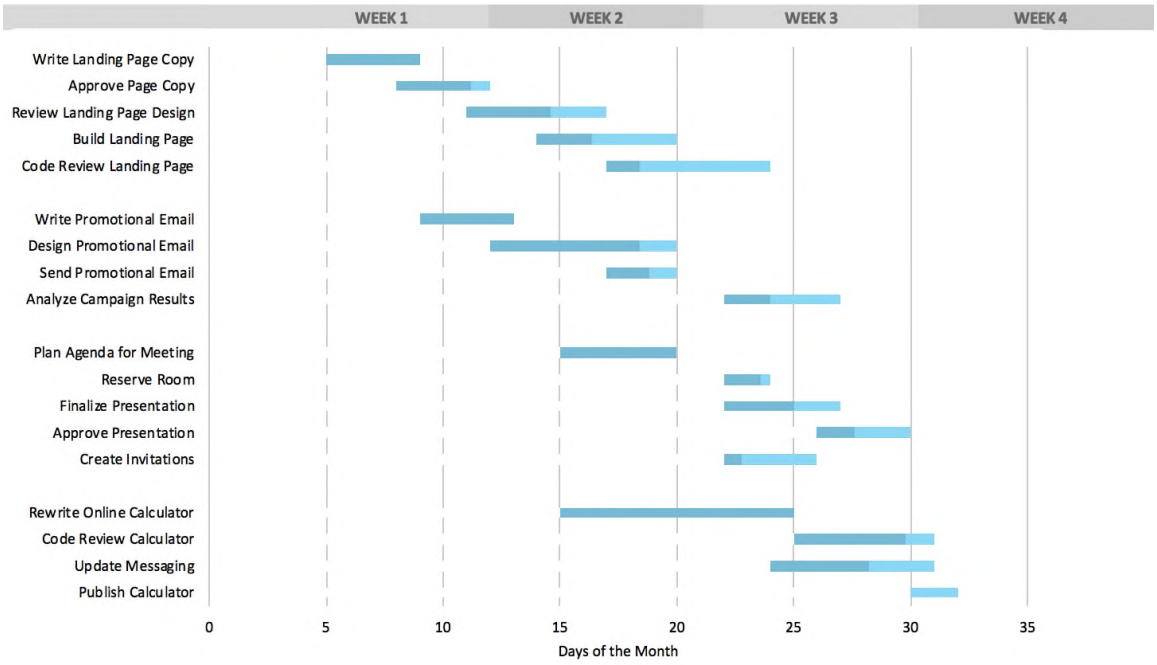


Рис. 1.2. Приклад діаграми Ганта

– табличний (рис. 1.3) (в підсумковій таблиці відображаються залежності між часом, виконавцями, роботами та інше);

День тижня	Початок заняття	Предмет	Клас	Ауд.	Вчитель
Понеділок	16.15	Історія України	11-А,Д	214	Березюк Г.В.
	17.30	Англійська мова	11-Г,Б	308	Голуб Л.З.
	18.00	Фізика	11-А,В	221	Дуляницька Л.К.
	18.30	Біологія	11-А,Б,В	128	Сарницька Г.О.
Вівторок	16.00	Історія України	11-Г	214	Березюк Г.В.
	16.00	Математика	11-Б,В,Г	125	Волько Л.І.
	17.30	Українська мова	11-А	310	Діжурко К.В.
	17.30	Українська мова	11-Г	314	Городна О.З.
	18.00	Українська мова	11-Б	315	Слотницька І.В.
	18.00	Фізика	11-Д (ЖВІ)	221	Полюхович О.В.
Середа	17.00	Українська мова	11-Д	317	Міськевич Л.П.
	16.00	Математика	11-А,Г,В	217	Онісковець Г.І.
	17.30	Географія	11-Б,В,Г	217	Огрода В.З.
Четвер	16.00	Історія України	11-В	214	Березюк Г.В.
	18.00	Українська мова	11-В	314	Дядечко Р.П.
	17.00	Хімія	11-А,Б.	316	Муравинець Л.М.

Рис. 1.3. Приклад табличного представлення розкладу

– векторний (рис. 1.4) (основним завданням є відображення порядку виконання робіт).

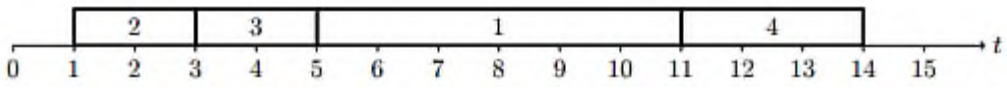


Рис. 1.4. Приклад векторного представлення розкладів

1.6. Висновки до розділу

В ході написання даного розділу були виконані наступні пункти:

– Проведений аналіз актуальності розроблюваної системи;
– Досліджена історія теорії розкладу;
– Проведено аналіз існуючих рішень;
– Визначено способи подання розкладів;
– Виділено ряд можливих вимог до розроблюваної системи.

Розглянута проблема складання розкладів у повсякденному житті в цілому та у роботі навчальних закладів в особливості. У цей напрям включені завдання призначення і розподілу навчального навантаження студентів і викладачів – складання розкладу занять. Ця складова навчального процесу регламентує трудовий ритм і впливає на творчу віддачу викладачів і студентів, тому її можна розглядати як фактор оптимізації використання обмежених трудових ресурсів.

Розглянуте питання актуальності теми. Було з’ясовано, що ручне складання розкладу має свої недоліки через людський фактор. Мозок людини не здатний з тією ж легкістю як комп'ютер обробляти велику кількість інформації. Подібна робота довіряється тільки досвідченим диспетчерам, але і вони не завжди здатні скласти оптимальний варіант розкладу. В даний час перед ВНЗами стоїть наступний вибір між тим, продовжувати складати графіки вручну, купувати ПЗ у сторонньої організації чи самостійно його розробляти.

Досліджено історію проблеми складання графіків. Було з’ясовано, що пошуки доказів *NP*-складності завдань теорії розкладів велися аж до 1991 року, коли нарешті був представлений перший теоретичний підхід для вивчення

обчислювальної складності в побудові наближених алгоритмів задач оптимізації, а також визначені класи цієї складності.

Аналіз існуючих рішень показав, що сьогоднішня ситуація на ринку така, що дуже мало програмних продуктів відповідає необхідній функціональності, яка потрібна для складання розкладу у ВНЗ. Такий стан речей легко пояснюється тим, що шкільна освіта на сьогоднішній день більш "стандартизована" (в сенсі організації навчального процесу), ніж вузівська. Така стандартизація веде до великого обсягу потенційного ринку продажів копій програмного забезпечення, що призводить до окупності продукту. Розглянуто такі програмні продукти, як *aSc Timetables (IBN), AVTOR-2* та «Методист».

Було з’ясовано, що на сьогоднішній день графіки в основному подаються у табличному, графічному та векторному вигляді.

РОЗДІЛ 2
ГЕНЕТИЧНИЙ АЛГОРИТМ

2.1. Історія появи еволюційних алгоритмів

Генетичні алгоритми є частиною більш загальної групи методів, які називають еволюційними обчисленнями, що поєднують різні варіанти використання еволюційних принципів для досягнення поставленої мети. Стимулом виникнення цих методів стали кілька відкриттів у біології. Чарльз Дарвін опублікував у 1859р. свою знамениту роботу "Походження видів", де були проголошені основні принципи еволюційної теорії: спадковість, мінливість і природний відбір. Тим самим було показано, що використання даних механізмів здатно привести до визначеного результату під впливом навколишнього середовища.

Однак довгий час залишалося відкритим питання про те, як же все-таки передається генетична інформація від батьків нащадкам. У 1944 році Эйвері, Маклеод і Маккарті опублікували результати своїх досліджень, що доводили, що за спадкоємні процеси відповідальна "кислота дезоксирибозного типу". Однак про те, як працює ДНК, стало відомо 27 квітня 1953 року, коли в номері журналу "*Nature*" вийшла стаття Уотсона і Лементу, які вперше запропонували модель дволанцюгової спіралі ДНК. У такий спосіб стали відомі всі необхідні компоненти для реалізації моделі еволюції на комп'ютері.

Тут варто трохи зупинитися на наступному. Справа в тім, що, напевно, єдина мета існування природного комплексу – виживання в постійно змінюваних умовах навколишнього середовища. І хоча сенс існування – споконвічне питання для філософів, але якщо трохи абстрагуватися від глобальних задумів начебто збагнення сутності Всесвіту, досягнення трансцендентності, злиття з космічним Я і пошуку відповіді на питання "Що було раніш, курка або яйце?", то можна зробити припущення, що сенс життя – у самому житті.

Загалом, можна сказати, що у всіх штучних системах є таке поняття, як призначення, інакше кажучи, зміст, мета. Ці параметри є й в еволюційних

алгоритмів, за винятком, може, систем штучного життя (*Artificial Life (A-Life) Systems*).

Перша публікація, яку можна віднести до генетичних алгоритмів з'явилася в 1957 році, автором був Барічеллі Н.А., і називалася вона "*Symbiogenetic evolution processes realised by artificial methods*". У 1962 році з'явилася ще одна його робота "*Numerical testing of evolution theories*". Приблизно в той же час ще один дослідник Фрейзер А.С. також опублікував дві статті: "*Simulation of genetic systems by automatic digital computers: S-linkage, dominance, and epistasis*" (1960) і "*Simulation of genetic systems*" (1962).

Незважаючи на те, що роботи обох авторів були спрямовані насамперед на розуміння природного феномена спадковості, робота Фрейзера має багато загального із сьогоднішнім баченням генетичних алгоритмів. Він моделював еволюцію 15-бітних рядків і підраховував процентний вміст особей із вдалим фенотипом в успішних поколіннях. Його роботи нагадують оптимізацію функцій, однак у роботах Фрейзера немає жодного згадування про можливості використовувати ГА для штучних задач. У багатьох джерелах саме Холланда називають батьком сучасної теорії ГА.

Однак, Холланд займався ними не із самого початку. Його цікавила, насамперед, здатність природних систем до адаптації, а його мрією було створення такої системи, що могла б пристосуватися до будь-яких умов навколишнього середовища. Заслуга Холланда в тім, що він усвідомив значення еволюційних принципів в адаптації і розвив свої припущення. Разом зі своїми студентами, що слухали курси по адаптивних системах в Університеті штату Мічиган, він розробляє те, що згодом назве "генетичним планом", а нам це відомо як "генетичний алгоритм". Про те, наскільки серйозно велися роботи в цьому напрямку говорить уже те, що один зі студентів Холланда – Кеннет Де Йонг захищав дисертацію на степінь Доктора Філософії у 1975 році. Дисертація називається "*An Analysis of the Behavior of a Class of Genetic Adaptive Systems*" і може стати першим путівником для тих, хто хоче займатися ГА. У тому ж році виходить знаменита книга Холланда

"Адаптація в природних і штучних системах" ("*Adaptation in Natural and Artificial Systems*", 1975).

Після цього ГА починають привертати до себе усе більше уваги, ними займається усе більше дослідників, що знаходять їм нові сфери застосування. У 1992 році, за даними бібліографії Ярмо Аландера, число публікацій перевалило за 500, що, узагалі ж, небагато, однак у 1982 році їх було всього 15.

На закінчення можна сказати, що ГА разом зі штучними нейронними мережами є напрямками сучасного штучного інтелекту. Справа в тім, що є ще класичний штучний інтелект, на якому побудовані більшість експертних систем і баз знань, його відмінна риса в тім, що він, в основному, використовує логіку для побудови висновків.

2.2. Природний відбір

Еволюційна теорія стверджує, що кожен біологічний вид цілеспрямовано розвивається і змінюється для того, щоб щонайкраще пристосуватися до навколишнього середовища. У процесі еволюції багато видів комах і риб придбали захисне фарбування, їжак став невразливим завдяки голкам, людина стала власником дуже складної нервової системи. Можна сказати, що еволюція – це процес оптимізації всіх живих організмів.

Основний механізм еволюції – це природний відбір. Його суть полягає в тому, що більш пристосовані особи мають більше можливостей для виживання і розмноження і, отже, приносять більше нащадків, ніж погано пристосовані особи. При цьому завдяки передачі генетичної інформації (генетичному спадкуванню) нащадки успадковують від батьків основні їхні якості. Таким чином, нащадки сильних індивідуумів також будуть відносно добре пристосованими, а їхня частка в загальній масі особин буде зростати. Після зміни декількох десятків або сотень поколінь середня пристосованість особин даного виду помітно зростає.

Щоб зробити зрозумілими принципи роботи генетичних алгоритмів, потрібно також пояснити, як улаштовані механізми генетичного спадкування в природі. У

кожній клітці будь-якої тварини утримується вся генетична інформація цієї особи. Ця інформація записана у виді набору дуже довгих молекул ДНК (Дезоксирибо Нуклеїнова Кислота). Кожна молекула ДНК – це ланцюжок, що складається з молекул нуклеотидів чотирьох типів, що позначаються *A*, *T*, *C* і *G*. Власне, інформацію несе порядок проходження нуклеотидів у ДНК. Таким чином, генетичний код індивідуума – це просто дуже довгий рядок символів, де використовуються всього 4 букви. У тваринній клітці кожна молекула ДНК оточена оболонкою – таке утворення називається хромосомою.

Кожна уроджена якість особи (колір ока, спадкоємні хвороби, тип волосся і т.д.) кодується визначеною частиною хромосоми, що називається геном цієї властивості. Наприклад, ген кольору ока містить інформацію, що кодує визначений колір очей. Різні значення гена називаються його аллелями.

При розмноженні тварин відбувається злиття двох батьківських полових кліток і їхні ДНК взаємодіють, утворюючи ДНК нащадка. Основний спосіб взаємодії – кроссовер (*cross-over*, схрещування). При кроссовері ДНК предків розділяються на дві частини, а потім обмінюються своїми половинками.

При спадкуванні можливі мутації через радіоактивність або інші впливи, у результаті яких можуть змінитися деякі гени в полових клітках одного з батьків. Змінені гени передаються нащадкові і додають йому нові властивості. Якщо ці нові властивості корисні, вони, швидше за все, збережуться в даному виді – при цьому відбудеться стрибкоподібне підвищення пристосованості виду. Ключову роль в еволюційній теорії грає природний відбір. Його суть полягає в тому, що найбільш пристосовані особи краще виживають і приносять більше нащадків, чим менш пристосовані. Сам по собі природний відбір ще не забезпечує розвиток біологічного виду. Тому дуже важливо зрозуміти, яким чином відбувається спадкування, тобто як властивості нащадка залежать від властивостей батьків. Основний закон спадкування інтуїтивно зрозумілий – він полягає в тому, що нащадки схожі на батьків. Зокрема, нащадки більш пристосованих батьків будуть, швидше за все, одними з найбільш пристосованих у своєму поколінні. Щоб зрозуміти, на чому заснована ця подібність, потрібно трохи поглибитися в побудову природної клітки –

у світ генів і хромосом. Майже в кожній клітці будь-якої особи є набір хромосом, що несуть інформацію про цю особу. Основна частина хромосоми – нитка ДНК, що визначає, які хімічні реакції будуть відбуватися в даній клітці, як вона буде розвиватися і які функції виконувати.

Ген – це відрізок ланцюга ДНК, відповідальний за визначену властивість особи, наприклад за колір очей, тип волосся, колір шкіри і т.д.

Розрізняють два види кліток: полові (такі, як сперматозоїд і яйцеклітина) і соматичні. У кожної соматичній клітці людини утримується 46 хромосом. Ці 46 хромосом – насправді 23 пари, причому в кожній парі одна з хромосом отримана від батька, а друга – від матері. Парні хромосоми відповідають за однакові ознаки – наприклад, батьківська хромосома може містити ген чорного кольору ока, а парна їй материнська – ген блакитного кольору. Існують визначені закони, що керують участю тих або інших генів у розвитку особи. Зокрема, у нашому прикладі нащадок буде чорнооким, оскільки ген блакитних очей є "слабким" і придушується геном будь-якого іншого кольору.

У статевих клітинах хромосом тільки 23, і вони непарні. При заплідненні відбувається злиття чоловічих і жіночої статевих клітин і виходить клітина зародка, що містить 46 хромосом. Які властивості нащадок одержить від батька, а які – від матері? Це залежить від того, які саме статеві клітини брали участь у заплідненні. Процес вироблення статевих клітин (так називаний мейоз) в організмі піддається ймовірностям, завдяки яким нащадки багато в чому відрізняються від своїх батьків. При мейозі, відбуваються наступне: парні хромосоми соматичної клітки зближаються впритул, потім їхні нитки ДНК розриваються в декількох випадкових місцях і хромосоми обмінюються своїми частинами (рис. 2.1).

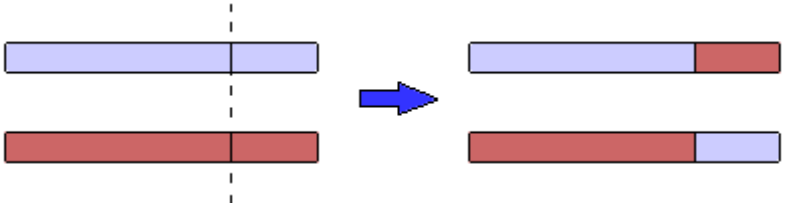


Рис. 2.1. Процес кросоверу

Цей процес забезпечує появу нових варіантів хромосом і називається "кросінговер". Кожна із знову створених хромосом виявиться потім усередині однієї з полових кліток, і її генетична інформація може реалізуватися в нащадках даної особи.

Другий важливий фактор, що впливає на спадковість, – це мутації, що виражаються в зміні деяких ділянок ДНК. Мутації також випадкові і можуть бути викликані різними зовнішніми факторами, такими, як радіоактивне опромінення. Якщо мутація відбулася в половій клітці, то змінений ген може передатися нащадкові і проявитися у виді спадкоємної хвороби або в інших нових властивостях нащадка. Вважається, що саме мутації є причиною появи нових біологічних видів, а кросінговер визначає вже мінливість усередині виду (наприклад, генетичні розходження між людьми).

2.3. Що таке генетичний алгоритм

Генетичний алгоритм - це метод вирішення як обмежених, так і необмежених задач оптимізації, який базується на природному відборі, процесі, що керує біологічною еволюцією. Генетичний алгоритм неодноразово модифікує сукупність окремих рішень. На кожному кроці генетичний алгоритм відбирає випадкових людей із поточної популяції для батьків і використовує їх для виробництва дітей для наступного покоління. Протягом наступних поколінь населення «еволюціонує» до оптимального рішення.

Ви можете застосувати генетичний алгоритм для вирішення різноманітних задач оптимізації, які погано підходять для стандартних алгоритмів оптимізації, включаючи задачі, в яких цільова функція є розривною, недиференційованою, стохастичною або вкрай нелінійною. Генетичний алгоритм може вирішувати задачі змішаного цілочисельного програмування, де деякі компоненти обмежуються цілочисельним значенням.

Нехай дана деяка складна функція (цільова функція), що залежить від декількох змінних, і потрібно знайти такі значення змінних, при яких значення

функції максимальне. Задачі такого роду називаються задачами оптимізації і вони зустрічаються на практиці дуже часто.

Один з найбільш наочних прикладів – задача розподілу інвестицій. У цій задачі змінними є обсяги інвестицій у кожен проект (10 змінних), а функцією, яку потрібно максимізувати – сумарний дохід інвестора. Також приведені значення мінімального і максимального об’єму вкладення в кожний із проектів, що задають область зміни кожної із змінних.

Спробуємо розв’язати цю задачу, застосовуючи відомі нам природні способи оптимізації. Будемо розглядати кожен варіант інвестування (набір значень змінних) як індивідуум, а прибутковість цього варіанта – як пристосованість цього індивідуума. Тоді в процесі еволюції (якщо ми зуміємо його організувати) пристосованість індивідуумів буде зростати, а виходить, будуть з'являтися усе більш і більш дохідні варіанти інвестування. Зупинивши еволюцію в деякий момент і вибравши найкращого індивідуума, ми одержимо досить гарне рішення задачі.

Генетичний алгоритм – це проста модель еволюції в природі, яка реалізована у виді комп'ютерної програми.

У ньому використовуються як аналог механізму генетичного спадкування, так і аналог природного відбору. При цьому зберігається біологічна термінологія в спрощеному виді. От як моделюється генетичне спадкування (табл. 2.1).

Таблиця 2.1

Спрощена термінологія генетичного алгоритму

Хромосома	Вектор (послідовність) з нулів і одиниць. Кожна позиція (біт) називається геном.
Індивідуум – генетичний код	Набір хромосом – варіант рішення задачі
Кросовер	Операція, при якій дві хромосоми обмінюються своїми частинами
Мутація	Випадкова зміна однієї або декількох позицій в хромосомі

Щоб змоделювати еволюційний процес, спочатку згенеруємо випадкову популяцію – кілька індивідуумів з випадковим набором хромосом (числових

векторів). Генетичний алгоритм імітує еволюцію цієї популяції як циклічний процес схрещування індивідуумів і зміни поколінь (рис. 2.2).

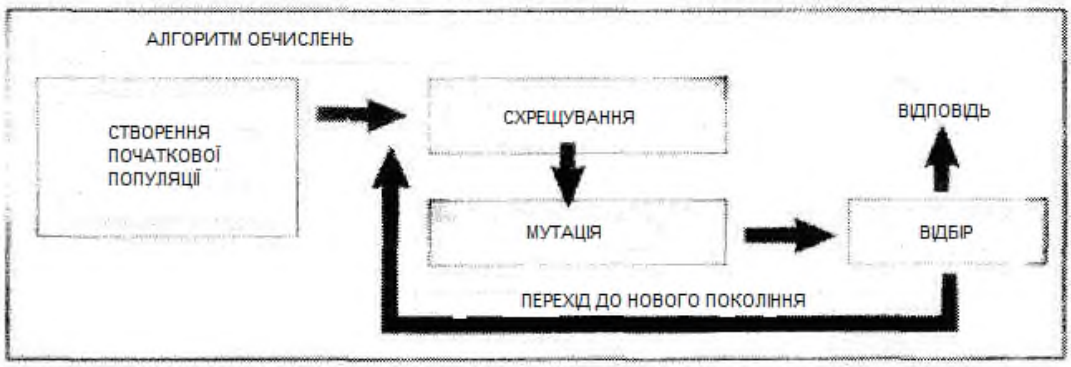


Рис. 2.2. Принцип роботи генетичного алгоритму

Життєвий цикл популяції – це кілька випадкових схрещувань (за допомогою кроссовера) і мутацій, у результаті яких до популяції додається якась кількість нових індивідуумів. Відбір у генетичному алгоритмі – це процес формування нової популяції зі старої, після чого стара популяція гине. Після відбору до нової популяції знову застосовуються операції кросовера і мутації, потім знову відбувається відбір, і так далі.

Відбір у генетичному алгоритмі тісно зв'язаний із принципами природного відбору в природі (табл. 2.2).

Таблиця 2.2

Зв'язок генетичного алгоритму з природним відбором

Пристосування індивідуума	Значення цільової функції на цьому індивідуумі
Виживання найбільш пристосованих	Популяція наступного покоління формується у відповідності з цільовою функцією. Чим пристосованіший індивідуум, тим більша ймовірність його участі у кросовері, тобто розмноженні

Таким чином, модель відбору визначає, яким чином варто будувати популяцію наступного покоління. Як правило, ймовірність участі індивідуума в схрещуванні береться пропорційно його пристосуванню. Часто використовується так називана стратегія елітизма, при якій кілька кращих індивідуумів переходять у наступне

покоління без змін, не беручи участі у кросовері і доборі. У будь-якому випадку кожне наступне покоління буде в середньому краще попереднього. Коли пристосованість індивідуумів перестає помітно збільшуватися, процес зупиняють і як розв’язання задачі оптимізації беруть найкращого зі знайдених індивідуумів.

2.4. Особливості генетичних алгоритмів

Генетичний алгоритм – новітній, але не єдино можливий спосіб розв’язання задач оптимізації. Віддавна відомі два основних шляхи розв’язання таких задач – переборний і локально-градієнтний. У цих методів свої переваги і недоліки, і в кожному конкретному випадку варто подумати, який з них вибрати. Розглянемо переваги і недоліки стандартних і генетичних методів на прикладі класичної задачі комівояжера (*TSP - travelling salesman problem*). Суть задачі полягає в тому, щоб знайти найкоротший замкнений шлях обходу декількох міст, заданих своїми координатами. Виявляється, що вже для 30 міст пошук оптимального шляху являє собою складну задачу, що спонукала розвиток різних нових методів (у тому числі нейромереж і генетичних алгоритмів).

Кожен варіант розв’язання (для 30 міст) – це числовий ряд, де на *j*-ому місці стоїть номер *j*-ого по порядку обходу міста. Таким чином, у цій задачі 30 параметрів, причому не всі комбінації значень припустимі. Природно, першою ідеєю є повний перебір усіх варіантів обходу.

Переборний метод найбільш простий по своїй суті і тривіальний у програмуванні. Для пошуку оптимального розв’язання (точки максимуму цільової функції) потрібно послідовно обчислити значення цільової функції у всіх можливих точках, запам'ятовуючи максимальне з них (рис. 2.3).

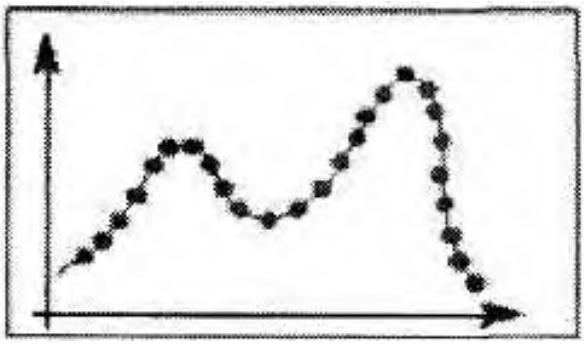


Рис. 2.3. Приклад можливої функції переборного методу

Недоліком цього методу є велика обчислювальна вартість. Зокрема, у задачі комівояжера буде потрібно прорахувати довжини більш 10^{30} варіантів шляхів, що зовсім нереально. Однак, якщо перебір усіх варіантів за розумний час можливий, то можна бути абсолютно упевненим у тім, що знайдене розв’язання дійсно оптимальне.

Другий популярний спосіб заснований на методі градієнтного спуска. При цьому спочатку вибираються деякі випадкові значення параметрів, а потім ці значення поступово змінюють, домагаючись найбільшої швидкості росту цільової функції (рис.2.4).

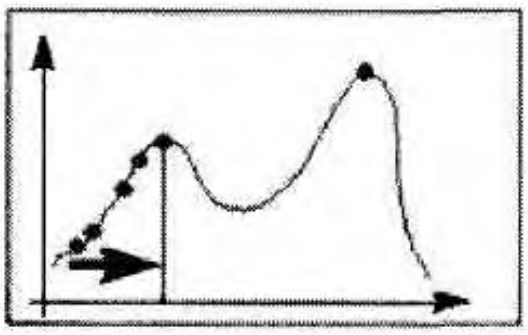


Рис. 2.4. Приклад можливої функції способу градієнтного спуску

Досягши локального максимуму, такий алгоритм зупиняється, тому для пошуку глобального оптимуму будуть потрібні додаткові зусилля.

Градієнтні методи працюють дуже швидко, але не гарантують оптимальності знайденого розв’язання.

Вони ідеальні для застосування в так званих унімодальних задачах, де цільова функція має єдиний локальний максимум (він же – глобальний). Легко бачити, що задача комівояжера унімодальною не є (рис. 2.5).

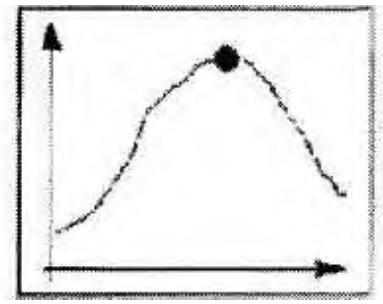


Рис. 2.5. Приклад можливої цільової функції задачі комівояжера

Типова практична задача, як правило, мультимодальна і багатомірна, тобто містить багато параметрів. Для таких задач не існує жодного універсального методу, що дозволяв би досить швидко знайти абсолютно точне розв’язання (рис. 2.6).

Рис. 2.6. Приклад можливої функції мультимодальної та багатомірної задачі

Однак, комбінуючи переборний і градієнтний методи, можна сподіватися отримати хоча б наближене розв’язання, точність якого буде зростати при збільшенні часу розрахунку (рис. 2.7).

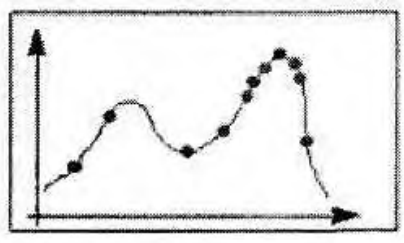


Рис. 2.7. Приклад можливої функції комбінації переборного і градієнтного методів

Генетичний алгоритм являє собою саме такий комбінований метод. Механізми схрещування і мутації в якомусь змісті реалізують переборну частину методу, а добір кращих розв'язань – градієнтний спуск. На рисунку показано, що така комбінація дозволяє забезпечити стійку, гарну ефективність генетичного пошуку для будь-яких типів задач (рис. 2.8).

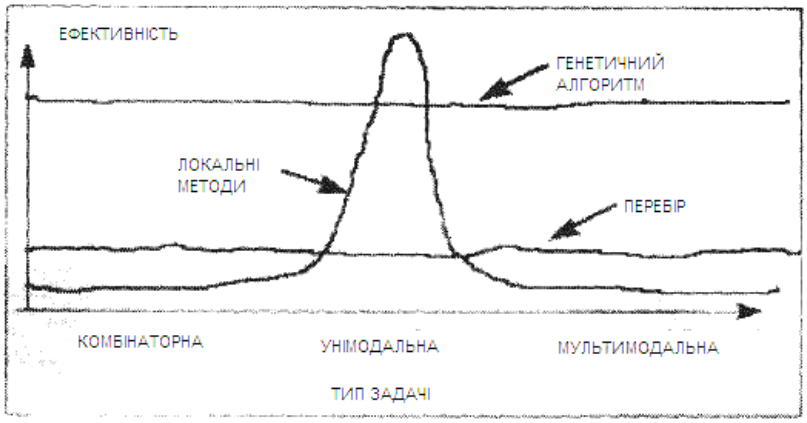


Рис. 2.8. Ефективність різних алгоритмів

Отже, якщо на деякій множині задана складна функція від декількох змінних, то генетичний алгоритм – це програма, що за розумний час знаходить точку, де значення функції досить близьке до максимально можливого. Вибираючи прийнятний час розрахунку, ми одержимо одне з кращих розв’язань, що узагалі можливо одержати за цей час.

2.5. Застосування генетичного алгоритму в задачах оптимізації

Генетичні алгоритми в основному застосовуються для розв’язання задач оптимізації, тобто задач, у яких є деяка функція декількох змінних $F(x_1, x_2, \ldots, x_n)$ і необхідно знайти або її максимум, або її мінімум. Функція F називається цільовою функцією, а змінні – параметрами функції. Генетичні алгоритми "пришиваються" до даної задачі в такий спосіб. Параметри задачі є генетичним матеріалом – генами. Сукупність генів складає хромосому. Кожна особа має свою хромосому, а, отже, свої набори параметрів.

Підставивши параметри в цільову функцію, можна одержати якесь значення. Те, наскільки це значення задовольняє поставленим умовам, визначає характеристику особи, що називається пристосованістю (*fitness*). Функція, що визначає пристосованість повинна задовольняти наступній умові: чим "краща" особа, тим вище пристосованість.

Генетичні алгоритми працюють з популяцією як правило фіксованого розміру, що складається з особин, заданих за допомогою способу, описаного вище. Особи "схрещуються" між собою за допомогою генетичних операторів, і в такий спосіб виходять нащадки, причому частина нащадків замінюють представників більш старого покоління у відповідності зі стратегією формування нового покоління.

Вибір особин для схрещування проводиться відповідно до селективної стратегії (*selection strategy*). Заново сформована популяція знову оцінюється, потім вибираються найбільш гідні для схрещування особи, що схрещуються між собою, виходять "діти" і займають місце "старих" індивідумів і т.д. Усі це продовжується доти поки не знайдеться особа, гени якої представляють оптимальний набір параметрів, при яких значення цільової функції близьке до максимуму або мінімуму, або дорівнює йому.

Зупинка роботи ГА може відбутися також у випадку, якщо популяція вироджується, тобто якщо практично немає розмаїтості в генах особин популяції, або якщо просто вийшов ліміт часу. Виродження популяції називають передчасною збіжністю (*premature convergence*).

Щоб застосувати генетичний алгоритм до проблеми планування, спочатку ми повинні представити його як геном. Один із способів представити геном планування - це визначити послідовність завдань та час початку цих завдань відносно один одного. Кожне завдання та відповідний час початку представляє ген.

Конкретна послідовність завдань і час початку (гени) представляє один геном у нашій популяції. Щоб переконатися, що наш геном є можливим рішенням, ми повинні подбати про те, щоб він відповідав нашим обмеженням пріоритету. Ми генеруємо початкову сукупність, використовуючи випадковий час початку в межах обмежень пріоритету.

Потім за допомогою генетичних алгоритмів ми беремо цю початкову популяцію та перетинаємо її, поєднуючи геноми разом з невеликою кількістю випадковості (мутації). Потомство цієї комбінації вибирається на основі функції фітнесу, яка включає одне або багато наших обмежень, таких як мінімізація часу та мінімізація дефектів. Ми дозволяємо цьому процесу тривати або заздалегідь відведений час, або поки ми не знайдемо рішення, яке відповідає нашим мінімальним критеріям.

Загалом кожне наступне покоління матиме більший середній рівень фізичної підготовки, тобто забиратиме менше часу з вищою якістю, ніж попередні покоління. Під час планування проблем, як і у випадку з іншими рішеннями генетичних алгоритмів, ми повинні переконатися, що ми не відбираємо потомство, яке є неможливим, наприклад, потомство, яке порушує наше обмеження пріоритету.

Звичайно, нам, можливо, доведеться додати додаткові показники придатності, такі як мінімізація витрат; однак кожне додане обмеження значно збільшує простір пошуку та зменшує кількість рішень, які добре відповідають.

Може створитися враження, що ГА є просто перекрученим варіантом випадкового пошуку. Але пристосованість була введена зовсім не даремно. Справа в тім, що вона безпосередньо впливає на шанс особи взяти участь у схрещуванні з наступним "народженням дітей". Вибираючи щораз для схрещування найбільш пристосованих особин, можна з визначеним степенем упевненості стверджувати, що нащадки будуть або не набагато гірші, ніж батьки, або кращі їх. Приблизно цю величину впевненості можна оцінити за допомогою теореми шаблонів (теореми шим).

Усього застосувань ГА дуже багато, тому приведений список не є вичерпним.

– Екстремальні задачі (пошук точок мінімуму і мінімуму);
– Задачі про найкоротший шлях;
– Задачі компонування;
– Складання розкладів;
– Апроксимація функцій;
– Добір (фільтрація) вхідних даних;

– Настроювання штучної нейронної мережі;
– Моделювання штучного життя (*Artificial life systems*);
– Біоінформатика (згортання білків і РНК);
– Ігрові стратегії;
– Нелінійна фільтрація;
– Агенти, що розвиваються/машини (*Evolvable agents/machines*);
– Різноманітні задачі на графах (задача комівояжера, розфарбування, знаходження паросполучень);
– Навчання штучної нейронної мережі;
– Штучне життя.

ГА - це ефективний метаевристичний інструмент для вирішення таких проблем управління, як проблема планування об'єкта, проектування мережі постачання, планування, прогнозування та управління запасами.

Д. Датта використовував ГА для вирішення однорядкової проблеми компонування об'єкта. Для цієї задачі модифіковані оператори кросоверу та мутації ГА дають дійсні рішення. Вони застосовували ГА до великих проблем, що складаються з 60–80 випадків. Однак він страждає від проблеми залежності параметрів.

Г. Садрзаде запропонував генетичний алгоритм для багаторядкової проблеми компонування об’єкта мати багато продуктів. Об’єкти були скупчені за допомогою мутаційних та евристичних операторів. Загальна вартість, отримана від запропонованого ГА, була зменшена на 7,2% порівняно з іншими алгоритмами. В. Ву впровадив ієрархічний ГА, щоб з’ясувати схему стільникової виробничої системи. Однак на результативність ГА сильно впливають генетичні оператори.

Т. Айелло запропонував МОГА для проблеми компонування об’єкта. Вони використовували мультиоб’єктний генетичний алгоритм для планування двадцяти різних відділів. Р. Паломо-Ромеро запропонував острівну модель ГА для вирішення.

Запропонована методика підтримує різноманітність популяцій та створює кращі рішення, ніж існуючі методи. Однак ця методика страждає від неправильної

міграційної стратегії, яка може бути використана для поліпшення населення. ГА та його варіанти успішно застосовані на проблемах компонування об’єкта.

Завдяки розвитку мультимедійних програм зображення, відео та аудіо передаються з одного місця в інше через Інтернет. У літературі було встановлено, що зображення частіше схильні до помилок під час передачі. Тому необхідні такі методи захисту зображення, як шифрування, водяні знаки та криптографія. Класичні методи шифрування зображень вимагають вхідних параметрів для шифрування.

Неправильний вибір вхідних параметрів призведе до неадекватних результатів шифрування. ГА та його варіанти були використані для вибору відповідних параметрів управління. М. Каур розробив багатоцільовий генетичний алгоритм для оптимізації контрольних параметрів хаотичної карти. Секретний ключ був створений за допомогою бета-хаотичної карти. Створений ключ використовувався для шифрування зображення. Паралельні ГА також використовувались для шифрування зображення.

Основними завданнями обробки зображень є попередня обробка, сегментація, виявлення об’єктів, знешкодження та розпізнавання. Сегментація зображень є важливим кроком для вирішення проблем обробки зображень. Розкладання та розділення зображення вимагає великих обчислювальних витрат.

Для вирішення цієї проблеми використовується ГА завдяки кращим можливостям пошуку. *Enhancement* - це техніка для покращення якості та контрастності зображення. Для аналізу даного зображення потрібна краща якість зображення. ГА використовувались для посилення природного контрасту та збільшення зображення.

Деякі дослідники працюють над гібридизацією грубого набору з адаптивним генетичним алгоритмом для злиття атрибутів шуму та кольору. ГА використовувались для усунення шуму від даного зображення. ГА може бути гібридизованим з нечіткою логікою, щоб погіршити шумне зображення. Техніка відновлення, заснована на ГА, може бути використана для усунення серпанку, туману та смогу з даного зображення. Виявлення та розпізнавання об’єктів є складною проблемою в задачі. Модель суміші Гауса забезпечує кращу

продуктивність під час процесу виявлення та розпізнавання. Параметри управління оптимізовані за допомогою ГА.

ГА демонструє чудові результати для вирішення таких завдань планування, як планування роботи в магазині, інтегроване планування та планування процесів, тощо. Щоб поліпшити ефективність у вищезазначених областях планування, дослідники розробили різні генетичні уявлення, генетичні оператори та гібридизували ГА з іншими методами.

Наприклад, організації, працівники яких можуть мати нерегулярний графік роботи, повинні впоратися з проблемою розподілу робочих змін для всіх робітників, щоб задовольнити прогнозований попит на роботу горизонту планування з мінімальними витратами, розглядаючи низку обмежень, що стосуються діючого законодавства, індивідуальні уподобання кожного працівника і так далі.

Лікарні, готелі, телефонні компанії, універмаги, тощо - приклади установ, у яких зазвичай присутні працівники з ненормованим графіком. Визначення неоптимального робочого графіку для вищезазначених установ є багатоваріантною, мультимодальною, комбінаторною задачею оптимізації, що включає багато обмежень.

Такі характеристики перешкоджають застосуванню звичайних методів оптимізації, які, ймовірно, потраплять у пастку локальних оптимумів. З іншого боку, в останні роки ми стали свідками появи еволюційних алгоритмів, зокрема генетичних алгоритмів, які були успішно застосовані до ряду проблем з деякими з вищезазначених характеристик.

Тут перша проблема обмеженого робочого графіку визначена для випадку, коли працівники класифікуються на кілька груп або категорій. Потім розробляється індивідуальний генетичний алгоритм до її рішення шляхом визначення витрат, пов'язаних з порушенням обмежень та шляхом розробки евристичних операторів, дія яких полягає у зменшенні значень витрати на порушення.

Результати моделювання підтверджують застосовність запропонованого методу до реального світу проблеми із плануванням робочого графіку.

Припустимо, що існує організація, яка має неоднорідну робочу силу, класифіковану за кількома працівниками групи або категорії, як показано на рис. 2.9.

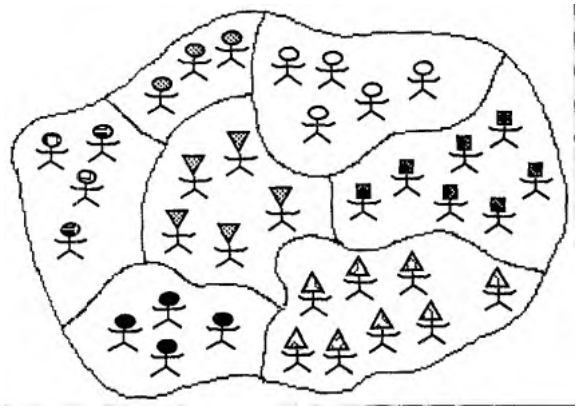


Рис. 2.9. Схематичне зображення різнорідної робочої сили з 36 працівників, розподілених на 7 груп або категорій

Запропонована проблема неоднорідного планування робочої сили із обмеженнями полягає у визначенні графіку роботи всіх робітників протягом даного періоду планування, щоб мінімізувати загальні витрати, задовольняючи при цьому кількість обмежень. Такі обмеження можуть бути загальними (наприклад, наявна кількість працівників), специфічна для групи (наприклад, максимальна та середня тривалість робочої зміни) або специфічна для працівника (наприклад, вподобання часу працівника або бажана кількість робочих змін).

Горизонт планування складається з дискретних, послідовних та рівномірних (однакової тривалості) інтервалів часу t, для $t = 1,2,3 ,. . . , T$. Кожен працівник працює по змінах, кожна зміна складається з цілого числа інтервалу часу. Крім того, нехай S позначає максимальну кількість робочих змін будь-якого працівника протягом горизонту планування.

Основні вимоги задачі включають R_t і R_t^g, відповідно, загальна кількість працівників та мінімальна кількість працівників групи g, необхідна на інтервалі t, для $t = 1,2 ,. . . , T$. Додатково є параметр у такий, що yR_t дає мінімально прийнятну підсумкову кількість працівників протягом інтервалу часу.

Іншим прикладом є розв’язання задачі комівояжера за допомогою генетичного алгоритму. Ця проблема є однією з найбільш інтенсивно вивчених проблем комбінаторної оптимізації, як приклад недетермінованих важких проблем з поліноміальним часом, з похідними у сферах логістики, кібер-медицини та медицини.

Задача задає наступне запитання: “Враховуючи перелік вузлів та відстані між кожною парою вузлів, який найкоротший шлях, який дозволяє відвідати кожен вузол лише один раз і повернутися до початкового вузла?”.

Кілька точних та евристичних алгоритмів можуть надати приблизне рішення, а час роботи алгоритмів залежить від кількості N вузлів.

Прикладом конструктивно-евристичного алгоритму $N!$ складності є алгоритм «Найближчий сусід», який пропонує маршрут із загальною відстанню, приблизно на 25% довшою за найкоротший можливий маршрут.

Такий алгоритм легко реалізувати, вибираючи найближчого сусіда в кожному вузлі, ігноруючи наслідки загальної відстані маршруту, що вказує на його слабкість. Таким чином, для отримання найкращого рішення необхідна оптимізація щонайменше 20% для такого алгоритму.

Для задачі комівояжера з N містами оптимальне рішення можна знайти методом “грубої сили” після проходження всіх різних $N!$ маршрутів, представлені N розмірним вектором цілих чисел.

Складність проблеми може різко зменшитися, якщо маршрут представити як ненаправлений гамільтонівський маршрут, що відповідає визначенню задачі. Це подання опускає значення напрямку і описує круговий маршрут (рис. 2.10) - таким чином складність падає з $N!$ до $(N-1)! / 2$ (рис. 2.11).

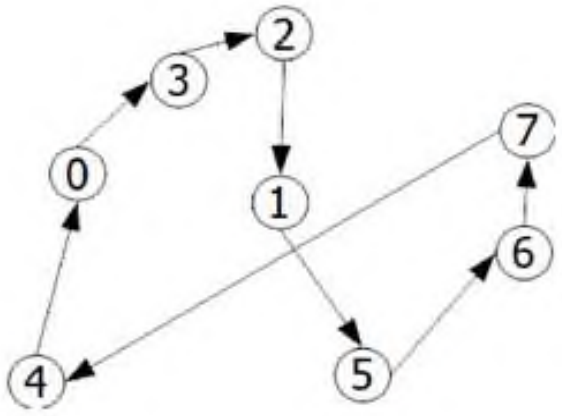


Рис. 2.10. Круговий маршрут, що складається з 8 вузлів, зображений у вигляді графа

[0 3 2 1 5 6 7 4], [4 7 6 5 1 2 3 0],
[3 2 1 5 6 7 4 0], [0 4 7 6 5 1 2 3],
[2 1 5 6 7 4 0 3], [3 0 4 7 6 5 1 2],
[1 5 6 7 4 0 3 2], [2 3 0 4 7 6 5 1],
[5 6 7 4 0 3 2 1], [1 2 3 0 4 7 6 5],
[6 7 4 0 3 2 1 5], [5 1 2 3 0 4 7 6],
[7 4 0 3 2 1 5 6], [6 5 1 2 3 0 4 7],
[4 0 3 2 1 5 6 7], [7 6 5 1 2 3 0 4]

Рис. 2.11. Той самий маршрут у векторному поданні

Для того, щоб розв’язати цю задачу за допомогою генетичного алгоритму, потрібно представити маршрут інакше. Можливе рішення (хромосома) - це вектор із відстанями між вузлами кожного можливого маршруту, який містить кожен вузол лише один раз і повертається до початкового вузла.

Практичний підхід, що випливає з теорії графів, використовує спрощену матрицю суміжності Зайделя (*SAM*) замість вектора з'єднання в парі з матрицею відстаней (*DM*).

SAM - це симетрична матриця [$N \times N$] бітів, де «0» означає розрив, а «1» - зв’язок між рядком і стовпчиком комірки (дугою між двома вузлами).

Для простих систем *SAM* може генеруватися безпосередньо з векторного подання. Для великих систем матрицю можна описати циклічно як 2-точковий КІП і

сформувати за допомогою простих матричних обчислень, які можна легко реалізувати в *Matlab*.

Безпосередньо помноживши верхній / нижній прямокутник матриці *SAM*, обчислюється загальна нормована відстань маршруту.

Критерієм припинення є найкоротший шлях (абсолютний мінімум) або коротший шлях, отриманий після максимальної кількості ітерацій.

Зазвичай початкова популяція особин генерується випадковим чином. Таким чином, частина початкових хромосом може бути далекою від оптимального рішення. Замість випадкового генерування використовується алгоритм *NN*, і початкові хромосоми («можливі рішення») є векторами, отриманими при розгляді кожного вузла як початкового міста на круговому маршруті (рис. 2.12). Таким чином, хромосоми - це шляхи із загальною відстанню приблизно на 25% довше, ніж найкоротший можливий шлях.

Chromosome 1 : 0 3 2 1 5 6 7 4
Chromosome 2 : 6 7 1 2 3 0 4 6

Рис. 2.12. Приклад двох початкових хромосом для маршруту

Відбір батьків базується на лінійному ранговому складі популяції, що включає m хромосом: $c_1, c_2, c_3 \ldots c_{m-2}, c_{m-1}, c_m$.

Мінімальне та максимальне рішення (c_1, c_m) - це перша пара батьків для наступного покоління. Щоб забезпечити двох додаткових батьків для наступного покоління, вдосконалений алгоритм використовує ортогональний-гамільтонів маршрут (*OHR*). Алгоритм використовує спеціальну функцію *OHR*: *OHR* відносно довгого маршруту створить коротший маршрут, а відносно короткого маршруту - довший. Таким чином, на першому етапі ітераційного процесу будуть визначені дві "сім'ї" - сім'я коротших маршрутів ("мінімальна" сім'я) і сім'я довших маршрутів ("максимальна" сім'я).

Застосовуючи *OHR*, алгоритм дозволить уникнути зменшення різноманітності населення, що є основною причиною передчасного зближення, а отже, забезпечить підтримку різноманітності.

Для великих груп населення можна розглянути більше пар батьків на прикладі: (c_1, c_m), (c_2, c_{m-1}), (c_3, c_{m-2}) тощо. Крім того, поділивши популяцію хромосом на k субпопуляцій, можна отримати k максимум і k мінімумів, і, отже, j-та пара батьків наступного покоління є j мінімумом і j максимумом, де $j = 1,2, \ldots k$. Додаткові батьки для наступного покоління забезпечуються за допомогою *OHR*.

Ітераційний процес визначається як два паралельних конвеєри (мінімальне та максимальне сімейство) з п’яти етапів на кожному. Процес для малих систем показано на рис. 2.13.

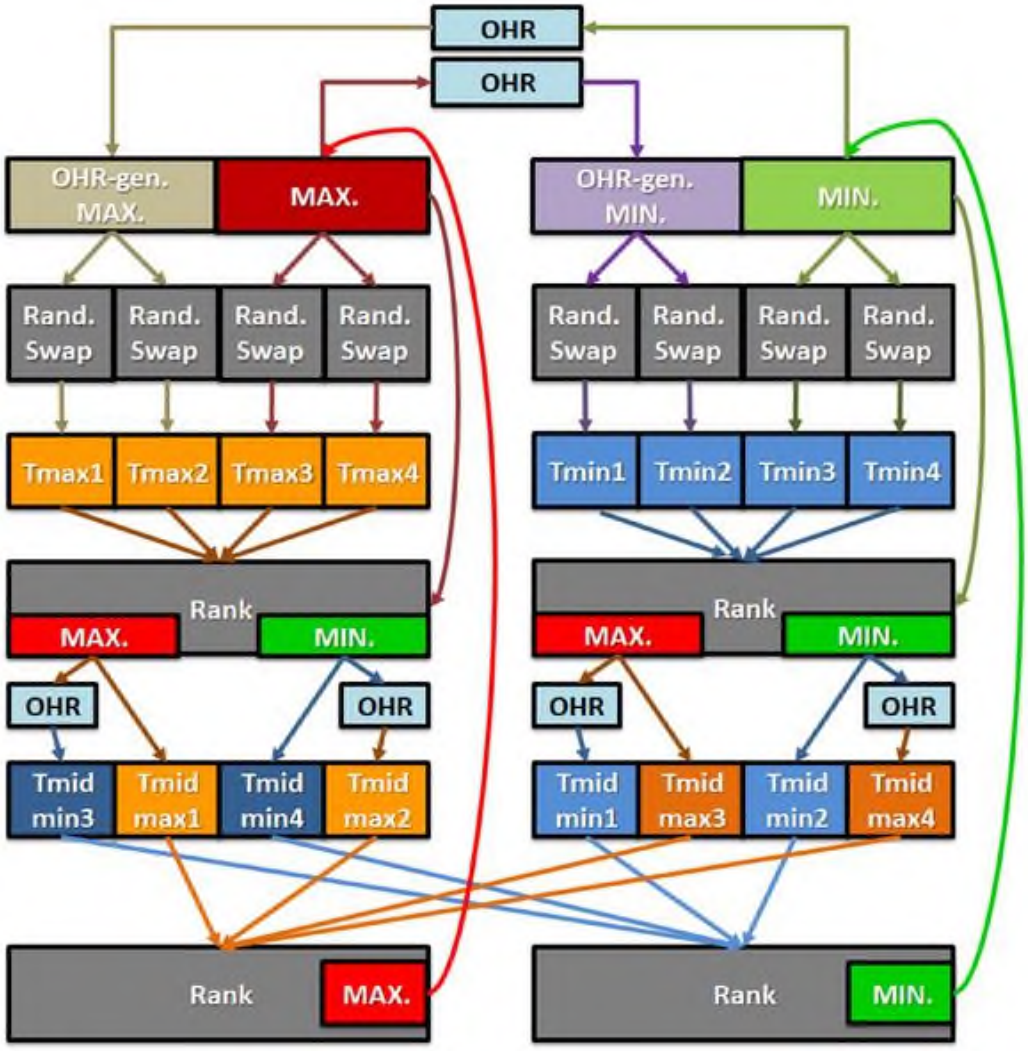


Рис. 2.13. Паралельне виконання алгоритму для найкоротших та найдовших маршрутів

На першому етапі, використовуючи функцію *OHR*, “мінімум” батьків надасть “відносно максимум” батьків, а “максимум” батьків має “відносно мінімум” батьків.

На другому етапі, використовуючи оператор мутації, “максимум” та “мінімум” батьків створюють сім’ю з чотирма T_{max} (тимчасовий максимум), відповідно сім’ю з чотирма хромосомами T_{min} (тимчасовий мінімум). На третьому етапі хромосоми сімейства «максимум» та «мінімум» класифікуються окремо.

Запропонована стратегія локальної оптимізації для виходу з локального мінімуму за допомогою циклічного вектора для підрахунку незмінних результатів алгоритму та ініціювання кросоверу та / або мутації з метою покращення рішення. На четвертому етапі, використовуючи функцію *OHR*, "максимум" і "мінімум" з кожної сім'ї дадуть нові "відносний мінімум" / "відносний максимум" хромосом (T_{midmin} *і* T_{midmax}).

На п’ятому етапі нові хромосоми будуть перенаправлені до відповідної сім’ї, і таким чином забезпечується нове ранговане покоління. Максимальна хромосома максимального сімейства та мінімальна хромосома мінімального сімейства - це перша пара батьків для наступного покоління, якщо критерій припинення ще не досягнутий.

Пропонується стратегія локальної оптимізації для виходу з локального мінімуму, використовуючи циклічний вектор для підрахунку кількості незмінних результатів та ініціювання кросинговеру та / або мутації з метою вдосконалення рішень. Кросовер заснований на трьох послідовних вузлах (з початковим випадковим розміщенням) кросовера порядку між двома батьками (рис. 2.14).

Parent 1	:	0 1 2 3 4 **5 6 7** 8 9
Parent 2	:	2 **5 6** *0 9 1 3 8 4* **7**
Offspring	:	2 *0 9 1 3* **5 6 7** *8 4*

Рис. 2.14. Зміна послідовності в результаті кросоверу

Операція мутації заснована на випадковому обміні парами вузлів. Підкачка може бути здійснена шляхом обміну їх рядків і після їх стовпців у *SAM* або шляхом заміни вузлів у стандартному векторі та створення відповідного *SAM*.

Псевдовипадковий двійковий генератор для кросоверу та мутації базується на 22-бітовому лінійному регістрі зсуву зворотного зв'язку (*LFSR*) з використанням поліноміального виразу:

$$1 + x^1 + x^2 + x^3 + x^5 + x^8 + x^{13} + x^{21}$$

Такий генератор виробляє (після вибору та нормалізації бітів) псевдовипадкові набори коефіцієнтів обміну бітів / кросовера.

2.6. Основні поняття генетичних алгоритмів

Генетичний алгоритм (англ. *Genetic algorithm*) – це евристичний алгоритм пошуку, використовуваний для рішення задач оптимізації і моделювання шляхом послідовного підбору, комбінування і варіації шуканих параметрів з використанням механізмів, що нагадують біологічну еволюцію. Є різновидом еволюційних числень (англ. *evolutionary computation*).

Відмінною рисою генетичного алгоритму є акцент на використання оператора "схрещування", що робить операцію рекомбінації рішень-кандидатів, роль якої аналогічна ролі схрещування в живій природі. "Батьком-засновником" генетичних алгоритмів, як вже було сказано вище, вважається Джон Холланд, книга якого "Адаптація в природних і штучних системах" (англ. *Adaptation in Natural and Artificial Systems*) є основною працею в цій області досліджень.

При описі генетичних алгоритмів використовуються означення, запозичені з генетики. Наприклад, мова йде про популяції особин, а як базові поняття застосовуються ген, хромосома, генотип, фенотип, аллель. Також використовуються відповідним цим термінам означення з технічного лексикону, зокрема, ланцюг, двійкова послідовність, структура.

Популяція – це кінцева множина особин.

Особи, що входять у популяцію, у генетичних алгоритмах представляються хромосомами з закодованим у них множинами параметрів задачі, тобто рішень, які інакше називаються точками в просторі пошуку (*search points*). У деяких роботах особи називаються організмами.

Хромосоми (інші назви – ланцюжки або кодові послідовності) – це упорядковані послідовності генів.

Ген (також названий властивістю, знаком або детектором) – це атомарний елемент генотипу, зокрема, хромосоми.

Генотип або структура – це набір хромосом даної особи. Отже, особами популяції можуть бути генотипи або одиничні хромосоми (у досить розповсюдженому випадку, коли генотип складається з однієї хромосоми).

Фенотип – це набір значень, що відповідають даному генотипові, тобто декодована структура або множина параметрів задачі (рішення, точка простору пошуку).

Аллель – це значення конкретного гена, також обумовлене як значення властивості або варіант властивості.

Локус або позиція вказує місце розміщення даного гена в хромосомі (ланцюжку). Множина позицій генів – це локи.

Дуже важливим поняттям у генетичних алгоритмах вважається функція пристосованості (*fitness function*), інакше називана функцією оцінки. Вона представляє міру пристосованості даної особи в популяції. Ця функція відіграє найважливішу роль, оскільки дозволяє оцінити степінь пристосованості конкретних особин у популяції і вибрати з них найбільш пристосовані (тобто найбільші значення функції, що мають пристосованості) відповідно до еволюційного принципу виживання "найсильніших" (найкраще пристосувалися).

Функція пристосованості також одержала свою назву безпосередньо з генетики. Вона впливає на функціонування генетичних алгоритмів і повинна мати точне і коректне означення. У задачах оптимізації функція пристосованості, як правило, оптимізується (точніше кажучи, максимізується) і називається цільовою

функцією. У задачах мінімізації цільова функція перетворюється, і проблема зводиться до максимізації.

У теорії керування функція пристосованості може приймати вид функції похибки, а в теорії ігор – вартісної функції. На кожній ітерації генетичного алгоритму пристосованість кожної особи даної популяції оцінюється за допомогою функції пристосованості, і на цій основі створюється наступна популяція особин, що складають множину потенційних рішень проблеми, наприклад, задачі оптимізації.

Чергова популяція в генетичному алгоритмі називається поколінням, а до знову створюваної популяції особин застосовується термін "нове покоління" або "покоління нащадків".

2.7. Опис алгоритму

Задача кодується таким чином, щоб її рішення могло бути представлене у виді вектора ("хромосома"). Випадковим чином створюється деяка кількість початкових векторів ("початкова популяція"). Вони оцінюються з використанням "функції пристосованості", у результаті чого кожному векторові привласнюється визначене значення ("пристосованість"), що визначає ймовірність виживання організму, представленого даним вектором.

Після цього з використанням отриманих значень пристосованості вибираються вектори (селекції), допущені до "схрещування". До цих векторів застосовуються "генетичні оператори" (у більшості випадків "схрещування"-*crossover* і "мутація"-*mutation*), створюючи в такий спосіб наступне "покоління". Особи наступного покоління також оцінюються, потім виробляється селекція, застосовуються генетичні оператори і т.д. Так моделюється "еволюційний процес", що продовжується кілька життєвих циклів (поколінь), поки не буде виконаний критерій зупинки алгоритму.

Таким критерієм може бути:

– Отримання глобального, або субоптимального рішення;

– Вичерпання числа поколінь, відпущених на еволюцію;

– Вичерпання часу, відпущеного на еволюцію.

Генетичні алгоритми служать, головним чином, для пошуку рішень у дуже великих, складних просторах пошуку.

Таким чином, можна виділити наступні етапи генетичного алгоритму (рис. 2.15):

– Створення початкової популяції;

– Обчислення функцій пристосованості для особин популяції (оцінювання).

Початок циклу:

– Вибір індивидів з поточної популяції (селекція);

– Схрещування і (або) мутація;

– Обчислення функцій пристосованості для всіх особин;

– Формування нового покоління;

– Якщо виконуються умови іншого, то кінець циклу, інакше на початок циклу.

Рис. 2.15. Схема алгоритму ГА

2.7.1. Створення початкової популяції

Перед першим кроком потрібно випадковим чином створити якусь початкову популяцію; навіть якщо вона виявиться зовсім неконкурентоспроможною, генетичний алгоритм досить швидко переведе її в життєздатну популяцію. Таким чином, на першому кроці можна особливо не намагатися зробити занадто дуже пристосованих особин, досить, щоб вони відповідали форматові особин популяції, і на них можна було підрахувати функцію пристосованості (*Fitness*). Підсумком першого кроку є популяція H, що складається з N особин.

2.7.2. Відбір

На етапі відбору потрібно з усієї популяції вибрати визначену її частку, що залишиться "у живих" на цьому етапі еволюції. Є різні способи проводити відбір. Ймовірність виживання особи h повинна залежати від значення функції пристосованості *Fitness* (h). Сама частка, що вижила, зазвичай є параметром генетичного алгоритму, і її просто задають заздалегідь. За підсумками відбору з N особин популяції H повинні залишитися s особин, що ввійдуть у підсумкову популяцію H'. Інші особи гинуть.

2.7.3. Розмноження

Розмноження в генетичних алгоритмах зазвичай статеве – щоб зробити нащадка, потрібно декілька батьків; зазвичай, потрібно рівно два. Розмноження в різних алгоритмах визначається по-різному – воно, зазвичай, залежить від представлення даних. Головна вимога до розмноження – щоб нащадок або нащадки мали можливість успадкувати риси обох батьків, "змішавши" їх яким-небудь досить розумним способом. Узагалі говорячи, для того щоб провести операцію розмноження, потрібно вибрати $(1-s)p/2$ пар гіпотез з H і провести з ними розмноження, одержавши по два нащадка від кожної пари (якщо розмноження

визначене так, щоб давати одного нащадка, потрібно вибрати $(1-s)p$ пар), і додати цих нащадків у H'. У результаті H' буде складатися з N особин. Чому особи для розмноження звичайно вибираються з усієї популяції H, а не з елементів, що вижили на першому кроці, H_0 (хоча останній варіант теж має право на існування)? Справа в тім, що головна критика багатьох генетичних алгоритмів – недолік розмаїтості (*diversity*) в особах. Досить швидко виділяється один-єдиний генотип, що являє собою локальний максимум, а потім всі елементи популяції програють йому вибір, і вся популяція "забивається" копіями цієї особи. Є різні способи боротьби з таким небажаним ефектом; один з них – вибір для розмноження не самих пристосованих, але узагалі всіх особин.

Оператори кросоверу використовуються для створення нащадків шляхом поєднання генетичної інформації двох або більше батьків. Добре відомими операторами кросовера є одноточкові, двоточкові, *k*-точкові, рівномірні, частково узгоджені, порядкові, переваги, що зберігають кросовер, перетасовка, зменшений сурогат і цикл.

У кросовері з однією точкою вибирається випадкова точка кросовера. Генетична інформація двох батьків, яка перевищує цю точку, буде мінятися місцями між собою. На рис 2.16. показана генетична інформація після обміну. Це замінило бітові масиви обох батьків, щоб отримати нове потомство.

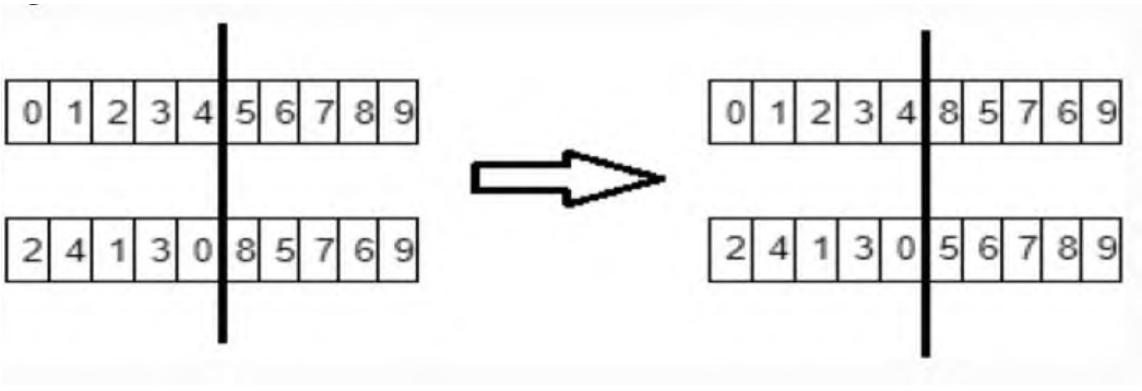


Рис. 2.16. Обмін генетичною інформацією у результаті одноточкового кросоверу

У двоточковому та *k*-точковому кросовері обираються дві або більше випадкових точок кросоверу, і генетична інформація батьків буде замінена

відповідно до створених сегментів. На рис. 2.17. показано обмін генетичної інформації між точками перетину. Середній сегмент батьків замінюється для створення нового потомства.

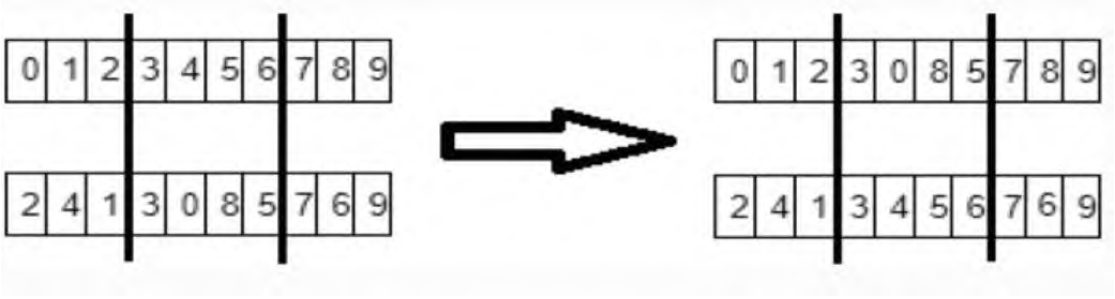


Рис. 2.17. Обмін генетичною інформацією у результаті двоточкового кросоверу

При рівномірному кросовері батьківський елемент не можна розкласти на сегменти. Батька можна розглядати як кожен ген окремо. Ми випадковим чином вирішуємо, чи нам потрібно поміняти ген таким же місцем розташування іншої хромосоми. На рис. 2.18. зображено обмін особами при рівномірній операції кросоверу.

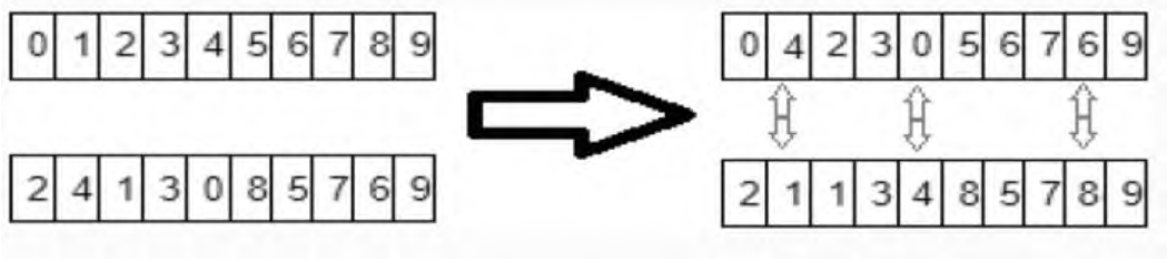


Рис. 2.18. Обмін інформацією через окремі гени

Частково узгоджений кросовер є найбільш часто використовуваним оператором кросовера. Це оператор, який працює ефективніше, ніж більшість інших операторів кросовера. Частково узгоджений кросовер був запропонований Д. Гольдбергом та Р. Лінглем. Для спарювання обирають двох батьків. Один з батьків передає якусь частину генетичного матеріалу, а відповідна частина іншого батька

знаходиться у нащадку. Після завершення цього процесу залишені алелі копіюються з другого батька. На рис. 2.19. зображено приклад такого кросоверу.

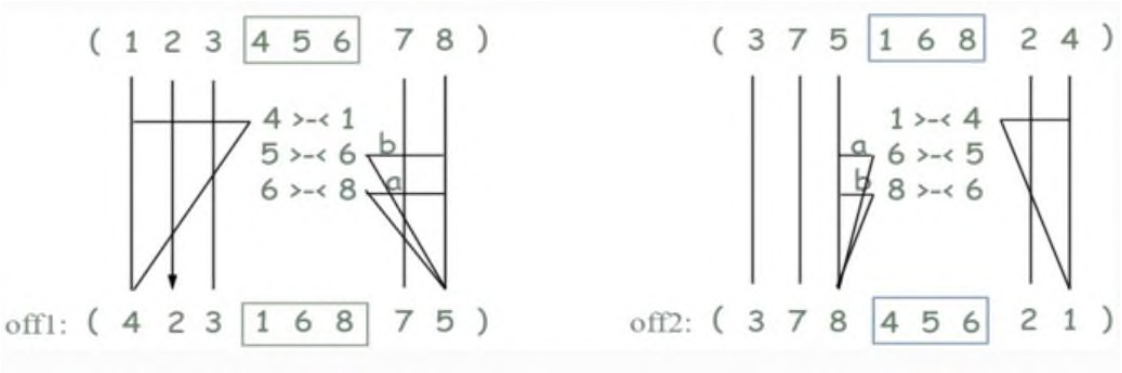


Рис. 2.19. Обмін інформацією під час частково узгодженого кросоверу

Впорядкований кросовер було запропоновано Девісом у 1985 році. Він копіює одну (або більше) частин батьківських даних нащадкам із вибраних точок вирізання та заповнює залишок простору значеннями, відмінними від тих, що включені в скопійований розділ. Варіанти такого кросоверу пропонуються різними дослідниками для різних типів проблем. Він добре підходить для задач впорядкування, однак виявлено, що він є менш ефективним у вирішенні задачі комівояжера.

Кросовер з збереженням переваги зберігає впорядкування окремих рішень, які є у батьківського потомства перед застосуванням кросоверу. Потомство ініціалізується на рядок випадкових 1 і 0, які вирішують, чи слід обирати осіб від обох батьків чи ні.

Кросовер із перестановками було запропоновано для зменшення упередженості, введеної іншими методами кросоверу. Він перемішує значення окремого рішення перед кросовером і відхилює їх після того, як виконана операція кросовера, так що точка кросовера не створює будь-яких упереджень у кросовері. Однак використання цього кросовера в останні роки дуже обмежене.

Сурогатний кросовер зі зниженням зменшує непотрібні кросовери, якщо батьки мають однакову послідовність генів для подання розчину. Він базується на припущенні, що генетичний алгоритм дає кращих особин, якщо батьки досить

різноманітні за своїм генетичним складом. Однак цей кросовер не може дати кращих особин для тих батьків, які мають однаковий склад.

Циклічний кросовер намагається створити потомство за допомогою батьків, де кожен елемент займає позицію, посилаючись на позицію своїх батьків. У першому циклі він бере деякі елементи у першого батька. У другому циклі він бере інші елементи з другого батька, як показано на рис. 2. 20.

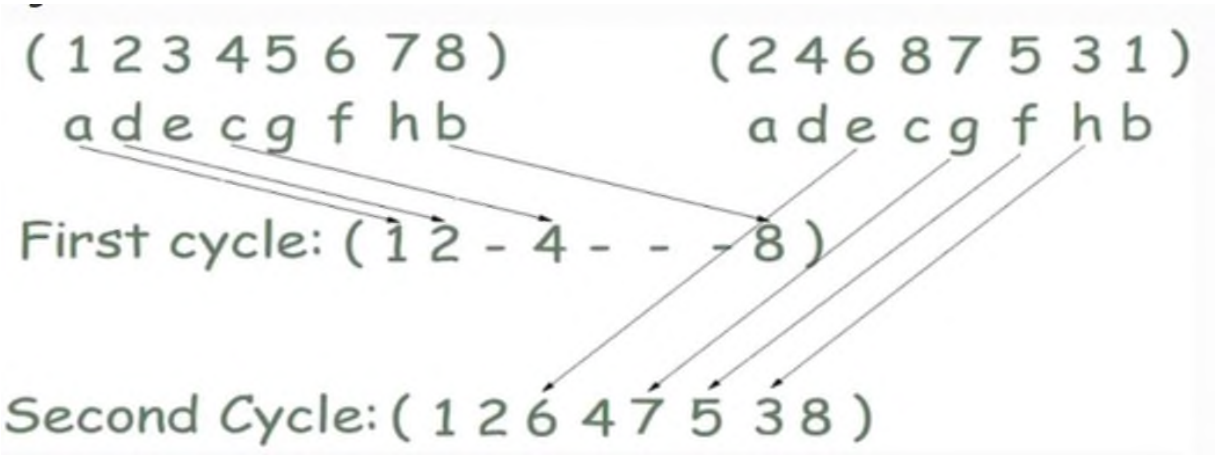


Рис. 2.20. Обмін інформацією у циклічному кросовері

Як показує практика, одноточкові та *k*-точкові типи кросоверу легко застосувати. Рівномірний кросовер підходить для великих підмножин. Впорядковані кросовери та циклічні забезпечують кращу розвідку, ніж інші техніки кросовера. Частково підібраний кросовер забезпечує краще дослідження. Ефективність частково узгодженого кросовера краща, ніж інші техніки кросовера. Зменшені сурогатні та циклічні кросовери страждають від передчасного зближення.

2.7.4. Мутації

До мутацій відноситься все те ж саме, що і до розмноження: є деяка частка мутантів m, що є параметром генетичного алгоритму, і на кроці мутацій потрібно вибрати m особин, а потім змінити їх відповідно до заздалегідь визначених операцій мутації.

Мутація - це оператор, який підтримує генетичне різноманіття від однієї популяції до наступної. Добре відомими операторами мутації є переміщення, проста інверсія та мутація із зіткненнями.

Оператор мутації переміщення витісняє підрядок даного індивідуального рядка всередині себе. Місце вибирається випадковим чином із даного підрядка для переміщення таким чином, щоб отримане рішення було дійсним, а також випадковою мутацією переміщення. Існують варіанти таких мутацій - це обмінна мутація та мутація із вставкою. В операторах мутації обміну та вставки мутації частина окремого рішення або обмінюється з іншою частиною, або вставляється в інше місце відповідно.

Простий оператор мутації інверсії змінює підрядок між будь-якими двома визначеними місцями в окремому рішенні. Це оператор інверсії, який перевертає випадково вибраний рядок і розміщує його у випадковому місці. Оператор мутації скремблінгу розміщує елементи у вказаному діапазоні індивідуального рішення у випадковому порядку та перевіряє, чи покращено значення придатності нещодавно створеного рішення чи ні.

Рівномірні та одноточкові кросовери можна використовувати з більшістю операторів кодування та мутації. Частково узгоджений кросовер використовується з мутацією інверсії, а схема кодування перестановки забезпечує оптимальне рішення.

2.8. Кодування

Для більшості обчислювальних проблем схема кодування (тобто перетворення в певній формі) відіграє важливу роль. Надана інформація повинна кодуватися у певному бітовому рядку. Схеми кодування диференційовані відповідно до проблемної області. Добре відомими схемами кодування є двійкові, вісімкові, шістнадцяткові, перестановки, засновані на значеннях і дерева.

Бінарне кодування - це загальновживана схема кодування. Кожен ген або хромосома представлені у вигляді рядка 1 або 0. У двійковому кодуванні кожен біт представляє характеристики рішення. Це забезпечує більш швидку реалізацію

операторів кросоверу та мутації. Однак для перетворення в двійкову форму потрібні додаткові зусилля, а точність алгоритму залежить від двійкового перетворення. Потік бітів змінюється відповідно до проблеми. Схема двійкового кодування не підходить для деяких проблем інженерного проектування через епістаз та природне уявлення.

У вісімковій схемі кодування ген або хромосома представлені у формі вісімкових чисел (0–7). У шістнадцятковій схемі кодування ген або хромосома представлені у вигляді шістнадцяткових чисел (0–9, *A*-*F*). Схема кодування перестановок, як правило, використовується для впорядкування завдань. У цій схемі кодування ген або хромосома представлені рядком чисел, що представляє положення в послідовності. У схемі кодування значень ген або хромосома представлені за допомогою рядка деяких значень. Ці значення можуть бути дійсними, цілими числами або символами. Ця схема кодування може бути корисною для вирішення проблем, в яких використовуються більш складні значення. Оскільки двійкове кодування може призвести до збою в таких проблемах, в основному він використовується в нейронних мережах для пошуку оптимальних ваг.

У кодуванні дерева ген або хромосома представлені деревом функцій або команд. Ці функції та команди можуть бути пов’язані з будь-якою мовою програмування. Це дуже схоже на подання репресій у деревному форматі. Цей тип кодування, як правило, використовується в програмах або виразах, що розвиваються.

2.9. Різновиди генетичних алгоритмів

Різні варіанти генетичних алгоритмів були запропоновані дослідниками. Варіанти ГА в цілому класифікуються на п’ять основних категорій, а саме реальне та двійкове кодування, багатооб’єктні, паралельні, хаотичні та гібридні генетичні алгоритми.

Виходячи з представлення хромосом, ГА класифікуються на два класи, а саме на двійкові та закодовані ГА.

Двійкове представлення було використано для кодування ГА і відоме як двійковий генетичний алгоритм. Генетичні оператори також були модифіковані для здійснення процесу пошуку. Пейн і Глен розробили двійковий ГА для виявлення подібності між молекулами. Вони використовували двійкове представлення для положення молекули та їх конформацій. Однак цей метод має високу обчислювальну складність.

В. Лонгян дослідив три різні методи для проектування вітроелектростанцій із використанням бінарного ГА (БГА). Їх метод дав кращу придатність і ефективність. Д. Шукла використовував БГА для вибору підмножини функцій. Він використовував концепцію максимізації взаємної інформації для вибору важливих ознак. БГА страждають від скель Хеммінга, нерівномірної схеми та труднощів у досягненні точності.

Закодовані ГА широко використовуються в різних програмах в повсякденному житті. Представлення хромосом тісно пов'язане з реальними проблемами. Основними перевагами таких ГА є надійність, ефективність та точність. Однак вони страждають від передчасного зближення. Дослідники працюють над ними, щоб поліпшити їх ефективність. Більшість закодованих ГА розробляються шляхом модифікації операторів кросоверу, мутації та відбору.

Мультиоб’єктний ГА (МОГА) - це модифікована версія простого ГА. МОГА відрізняються від ГА з точки зору призначення функції фітнесу. Решта кроків схожі на ГА. Основним мотивом мультиоб'єктного ГА є створення оптимального фронту Парето в об'єктному просторі таким чином, щоб подальше вдосконалення будь-якої функції фітнесу не порушувало інші функції фітнесу.

Конвергенція, різноманітність та покриття є основною метою багатоцільових ГА. МОГА в основному поділяються на дві категорії, а саме на основі Парето та на основі розкладання.

Поняття домінування Парето було введено в МОГА. Фонсека та Флемінг розробили перший МОГА. Для вирішення мультимодальних проблем було запропоновано концепцію ніші та осіб, що приймають рішення. Однак МОГА страждає від проблеми налаштування параметрів та ступеня тиску вибору. Горн

запропонував нішевий генетичний алгоритм Парето, який використовував концепцію відбору турнірів та домінування Парето. Шрінівас та Деб розробили генетичний алгоритм сортування без домінування. Однак він страждає від відсутності елітарності, необхідності спільного використання параметрів та високої складності обчислень.

Щоб полегшити ці проблеми, було розроблено швидкий елітарний генетичний алгоритм сортування без домінування. Його ефективність може погіршитися через багато об'єктивних проблем. Цей ГА не зміг зберегти різноманітність у Парето-фронті. Щоб полегшити цю проблему, ввели динамічну відстань скупчення і розробили мультиоб'єктний мікро-ГА. Вони використовували архів для зберігання нереалізованих рішень. Ефективність підходів, заснованих на Парето, може погіршитися у багатьох об'єктивних проблемах.

МОГА на основі декомпозиції розкладають задану задачу на безліч підзадач. Ці підзадачі вирішуються одночасно та обмінюються рішеннями між сусідніми підпроблемами. Ісібучі та Мурата розробили мультиоб'єктний генетичний локальний пошук (МОГЛП). У МОГЛП випадкові ваги використовувались для відбору батьків та місцевого пошуку їхніх нащадків. Вони використовували метод заміни поколінь та вибір рулетки. Яшкевич модифікував МОГЛП, використовуючи різні механізми відбору для батьків. Потім було запропоновано клітинний генетичний алгоритм для мультиоб'єктної оптимізації (К-МОГА), який був продовженням МОГА. Вони додали клітинну структуру в МОГА. У К-МОГА оператор відбору проводився на кожній сусідній комірці. К-МОГА було продовжено шляхом запровадження процедури імміграції та відомого як КІ-МОГА. Алвес та Алмейда розробили мультиоб'єктний генетичний алгоритм на основі Чебишева, що забезпечує зближення та різноманітність. Скалярна функція Чебишева була використана для створення набору рішень без домінування. Також було запропоновано МОГА на основі декомпозиції (Д-МОГА). Вони інтегрували навчання на основі опозиції в Д-МОГА для генерації векторів ваги. Д-МОГА здатний підтримувати баланс між різноманітністю рішень та дослідженням простору пошуку.

Метою паралельних ГА є покращення обчислювального часу та якості рішень за допомогою розподілених індивідуумів. Паралельні ГА класифікуються на три широкі категорії, такі як паралельні ГА ведучого-підлеглого, дрібнозернисті паралельні ГА та паралельний ГА з мультипопуляцією. У паралельному ГА ведучий-підлеглий обчислення функцій придатності розподіляється між декількома процесорами.

У дрібнозернистому ГА паралельні комп'ютери використовуються для вирішення реальних проблем. Генетичні оператори обмежені своїм сусідством. Однак взаємодія між індивідуумами дозволена. При грубозернистій ГА здійснюється обмін особинами між субпопуляціями. Параметри управління також передаються під час міграції. Основними проблемами паралельних ГА є максимізація пропускної здатності пам'яті та організація потоків для використання потужності графічних процесорів.

Велика кількість процесорів використовується в паралельному ГА ведучий-підлеглий у порівнянні з іншими підходами. Обчислення придатних функцій може бути збільшено за рахунок збільшення кількості процесорів. Такий ГА може бути використаний для вирішення проблем інтелектуального аналізу даних. Нечіткі правила використовуються з паралельним ГА. Оцінку придатності виконували на підлеглих машинах. Однак він страждає від великого обчислювального часу. Також, цей ГА реалізовано для проблеми пошуку шляху БПЛА. Генетичні оператори виконувались на процесорах. Вони використовували багатоядерний процесор з чотирма ядрами. Відбір та оцінку придатності проводили на машинах-підлеглих. Цей ГА було застосовано до проблеми присвоєння трафіку. Вони використали тридцять процесорів для вирішення цієї проблеми в Національному університеті Сінгапуру. Також існує веб-паралельний ГА, однак система має складний характер.

В останні кілька десятиліть дослідники працюють над міграційною політикою дрібнозернистого паралельного ГА (ДЗ-ПГА). Вони використовують годинник для частоти міграції, яка не залежить від поколінь і неоднорідну структуру та статичну конфігурацію. Для міграції було обрано найкраще рішення, а найгірше рішення замінено рішенням для мігрантів.

М. Курді використовував адаптивну частоту міграції. Процедура міграції починається до тих пір, поки не буде змін у отриманих рішеннях через десять послідовних поколінь. Використовували неоднорідну та динамічну структуру. Місцеві найкращі рішення були синхронізовані і сформували найкращі глобальні рішення. Найкращі глобальні рішення були передані всім процесорам для батьківського виконання. Частота міграції залежить від кількості генерації. Вони використовували єдину структуру з фіксованою конфігурацією.

Д. Чжан використовував паралельний ГА для вирішення проблеми покриття бездротових мереж. Вони застосовували стратегію «розділи і завоюй», щоб розкласти населення на субпопуляції. Після цього генетичні оператори застосовувались на місцевих розчинах, а Кун-Мюнкрес використовувався для об'єднання місцевих розчинів.

Ф. Пінель запропонував *GraphCell*. Популяція ініціалізувалася випадковими значеннями, а одне рішення ініціалізувалося евристичним методом *Min-min*. Для реалізації запропонованого підходу було використано 448 процесорів. Однак грубозернисті паралельні ГА менше використовуються через складний характер. Гібридні паралельні ГА широко використовуються в різних додатках.

А. Шаєгі запропонував Бірмінгемський кластеру ГА, заснований на пулі. Головний вузол відповідав за управління глобальною популяцією. Ведений вузол вибирав рішення з глобальної сукупності та виконував їх. Для обчислення використовується 240 процесорів.

В. Роберге використовував гібридний підхід для оптимізації кута перемикання інверторів. Вони використовували чотири різні стратегії для обчислення функції фітнесу. В наш час графічний процесор, «хмара» та мережа є найпопулярнішим обладнанням для паралельних ГА.

Основним недоліком ГА є передчасна конвергенція. Хаотичні системи включені в ГА для полегшення цієї проблеми. Різноманітність генетичного алгоритму хаосу усуває передчасну конвергенцію. Оператори кросовера та мутації можна замінити хаотичними картами. С. Тіонг інтегрував хаотичні карти в ГА для подальшого підвищення точності. Вони використовували шість різних хаотичних

карт. Продуктивність *Logistic, Henon* та *Ikeda* хаотичних ГА показує себе краще, ніж класичний ГА. Однак ці методи страждають від великої обчислювальної складності.

Р. Ебрагімзаде використовували хаос Лоренца для генетичних операторів ГА для усунення локальної проблеми оптимуму. Однак запропонований підхід не зміг знайти взаємозв'язок між ентропією та хаотичною картою. М. Хавіді використовували дві хаотичні карти, а саме логістичну карту та карту намету для генерування хаотичних значень замість випадкового відбору початкової сукупності.

Запропонований хаотичний ГА працює краще, ніж ГА. Однак цей метод страждає від високої обчислювальної складності. Г. Фуертес інтегрував ентропію в хаотичний ГА. Параметри управління модифікуються за допомогою хаотичних карт. Вони досліджували взаємозв'язок між ентропією та оптимізацією продуктивності.

Хаотичні системи також використовуються в мультиоб’єктних та гібридних ГА. Й. Або-Елага інтегрували хаотичну систему в модифіковану ГА для вирішення проблем дворівневого програмування. Хаотичність допомагає запропонованому алгоритму полегшити місцеві оптимуми та посилити конвергенцію. М. Тахір представив двійковий хаотичний ГА для вибору ознак у галузі охорони здоров’я.

Хаотичні карти використовувались для ініціалізації популяції, а модифіковані оператори відтворення застосовувались до популяції. Л. Сю запропонував хаотичний гібридний імунний ГА для розподілу спектра. Запропонований підхід використовує переваги як хаотичного, так і імунного оператора. Однак цей метод страждає від проблеми ініціалізації параметрів.

Генетичні алгоритми можна легко гібридизувати з іншими методами оптимізації для поліпшення їх продуктивності, такими як методи знешкодження зображення, оптимізація хімічних реакцій та багато інших. Основними перевагами гібридизованого ГА з іншими методами є краща якість результату, краща ефективність, гарантія можливих рішень та оптимізовані параметри управління.

З літератури видно, що на вибірковість ГА сильно впливає чисельність популяції. Щоб вирішити цю проблему, алгоритми локального пошуку, такі як меметичний алгоритм, болдвінівський, ламаркський та локальний пошук, були

інтегровані з ГА. Ця інтеграція забезпечує належний баланс між інтенсифікацією та диверсифікацією.

Ще однією проблемою в ГА є встановлення параметрів. Пошук відповідних параметрів управління - це нудне завдання. Інші метаевристичні методи можуть бути використані разом із ГА для вирішення цієї проблеми.

ГА були інтегровані з локальними алгоритмами пошуку для зменшення генетичного дрейфу. Явний оператор уточнення був введений у місцевий пошук для отримання кращих рішень. Т. Ель-Міхоуб встановив вплив ймовірності місцевого пошуку на чисельність популяції ГА. Ф. Еспіноза досліджував вплив місцевого пошуку на зменшення чисельності популяції ГА. Різні алгоритми пошуку інтегровані з ГА для вирішення реальних додатків.

У складних та багатовимірних задачах генетичні оператори ГА генерують неможливі рішення. Кросовер генерує нездійсненні рішення для проблем на основі замовлення. Оператор кросовера, що зберігає відстань, був розроблений для створення можливих рішень для задачі комівояжера. Оператор генофонду замість кросоверу був використаний для створення можливого рішення для кластеризації даних. А. Конак інтегрував алгоритм зрізу насиченості із ГА для проектування мереж зв'язку. Вони використовували рівномірний кросовер для отримання можливих рішень.

Існує можливість заміни генетичних операторів іншими методами пошуку. Л. Ленг розробив керований ГА, який використовує покарання з керованого місцевого пошуку. Ці покарання використовувались у фітнес-функції, щоб поліпшити показники ГА. А. Хедар використовував симплексний кросовер замість стандартного кросовера. Стандартний оператор мутації був замінений модельованим відпалом. Основні концепції квантових обчислень використовуються для підвищення продуктивності ГА. Евристичні оператори кросовера та «альпінізму» можуть бути інтегровані в ГА для вирішення проблеми з трьома збігами.

Параметри контролю ГА відіграють вирішальну роль у підтримці балансу між інтенсифікацією та диверсифікацією. Нечітка логіка має можливість оцінити

відповідні параметри управління ГА. Крім цього, ГА може бути використаний для оптимізації параметрів управління іншими методами.

ГА використовувались для оптимізації швидкості навчання, ваги та топології нейтральних мереж. ГА можна використовувати для оцінки оптимального значення нечіткого членства в контролері. Він також використовувався для оптимізації контрольних параметрів *ACO, PSO* та інших метаевристичних методів.

2.10. Висновки до розділу

У даному розділі було в загальному описано історію появи еволюційних алгоритмів, що таке генетичний алгоритм, його термінологію, особливості та етапи, з яких він складається. Для зрозумілості було побудовано блок-схему алгоритму та надано її опис. Також було визначено його переваги у задачах оптимізації.

Було з’ясовано, що генетичні алгоритми є частиною більш загальної групи методів, які називають еволюційними обчисленнями, що поєднують різні варіанти використання еволюційних принципів для досягнення поставленої мети. Стимулом виникнення цих методів стали кілька відкриттів у біології. Чарльз Дарвін опублікував у 1859р. свою знамениту роботу "Походження видів", де були проголошені основні принципи еволюційної теорії: спадковість, мінливість і природний відбір.

Загалом, термін «генетичний алгоритм» можна пояснити так: це метод вирішення як обмежених, так і необмежених задач оптимізації, який базується на природному відборі, процесі, що керує біологічною еволюцією. Генетичний алгоритм неодноразово модифікує сукупність окремих рішень. На кожному кроці генетичний алгоритм відбирає випадкових людей із поточної популяції для батьків і використовує їх для виробництва дітей для наступного покоління. Протягом наступних поколінь населення «еволюціонує» до оптимального рішення.

При описі генетичних алгоритмів використовуються означення, запозичені з генетики. Наприклад, мова йде про популяції особин, а як базові поняття застосовуються ген, хромосома, генотип, фенотип, аллель. Також використовуються

відповідним цим термінам означення з технічного лексикону, зокрема, ланцюг, двійкова послідовність, структура. Сформоване пояснення для кожного з термінів, перелічених вище та досліджено принципи роботи алгоритму.

Досліджено призначення генетичних алгоритмів у науці. Генетичні алгоритми в основному застосовуються для розв'язання задач оптимізації, тобто задач, у яких є деяка функція декількох змінних $F(x_1, x_2, \dots, x_n)$ і необхідно знайти або її максимум, або її мінімум. Функція F називається цільовою функцією, а змінні – параметрами функції. Генетичні алгоритми "пришиваються" до даної задачі в такий спосіб. Параметри задачі є генетичним матеріалом – генами. Сукупність генів складає хромосому. Кожна особа має свою хромосому, а, отже, свої набори параметрів.

З'ясовано, що основними етапами виконання алгоритму є формування початкової популяції, селекція, мутація, кросовер та обчислення функції пристосованості. Розглянуто особливості кожного з цих етапів.

Досліджено основні типи генетичних алгоритмів. З'ясовано, що ГА в цілому класифікуються на п'ять основних категорій, а саме реальне та двійкове кодування, мультиоб'єктні, паралельні, хаотичні та гібридні генетичні алгоритми. Розглянуто особливості алгоритмів кожної категорії.

На основі отриманих даних, було з'ясовано, що для реалізації програмного модулю необхідним і достатнім буде простий кодований генетичний алгоритм. Для реалізації програмного модулю було обрано мову програмування *Python* через наявність досить великої кількості розробників, що полегшить підтримку програмного модулю у майбутньому, а також можливість підключення бібліотек, написаних мовою *C* та прекомпіляцію коду, що дозволяє підвищити швидкість роботи програмного модулю.

РОЗДІЛ 3
ПРОЕКТУВАННЯ ПРОГРАМНОГО МОДУЛЯ

3.1. Постановка математичної моделі задачі складання графіку

Графік консультацій в університеті складається з таких множин: групи студентів, аудиторій, предметів, викладачів, часових інтервалів.

Множина груп студентів містить елементи, що представляють окремі групи:

$$G = \{g_1, g_2, \cdots, g_n\},$$

де g_i – це група;

n – кількість груп на курсі.

Кожен елемент цієї множини в свою чергу складається з чотирьох елементів:

– *id* групи;

– назва групи;

– розмір групи;

– множина предметів, що викладаються для цієї групи:

$$g_i = \{gid_i, gn_i, gs_i, gm_i\},$$

де gid_i – *id* групи;

gn_i – назва групи;

gs_i – розмір групи;

gm_i– множина предметів.

Множина предметів містить елементи, що представляють окремі предмети:

$$M = \{m_1, m_2, \cdots, m_n\},$$

де m_i – це предмет;

n – кількість предметів на курсі.

Кожен елемент цієї множини в свою чергу складається з чотирьох елементів:

– id предмета;

– шифр предмета;

– назва предмета;

– множина викладачів, що читають даний предмет.

$$m_i = \{mid_i, mc_i, mn_i, pids_i\},$$

де mid_i – id предмета;

mc_i – шифр предмета;

mn_i – назва предмета;

$pids_i$– множина викладачів.

Множина викладачів містить елементи, що представляють окремих викладачів:

$$P = \{p_1, p_2, \cdots, p_n\},$$

де p_i – це викладач;

n – кількість викладачів.

Кожен елемент цієї множини в свою чергу складається з двох елементів:

– id викладача;

– ПІБ викладача.

$$p_i = \{pid_i, pn_i\},$$

де pid_i – id викладача;

pn_i – ПІБ викладача.

Множина аудиторій, елементами якої є окремі аудиторії:

$$R = \{r_1, r_2, \cdots, r_n\},$$

де r_i – це аудиторія;

n – кількість аудиторій.

Кожен елемент цієї множини в свою чергу складається з трьох елементів:

– *id* аудиторії;

– номер аудиторії;

– місткість аудиторії (максимальна кількість студентів для навчання):

$$r_i = \{rid_i, rn_i, rc_i\},$$

де rid_i – *id* аудиторії;

rn_i – номер аудиторії;

rc_i – місткість аудиторії.

Множина часових інтервалів, елементами якої є окремі часові інтервали:

$$T = \{t_1, t_2, \cdots, t_n\},$$

де r_i – це часовий інтервал;

n – кількість часових інтервалів.

Кожен елемент цієї множини в свою чергу складається з двох елементів:

– *id* часового інтервалу;

– часовий проміжок:

$$t_i = \{tid_i, ts_i\},$$

де tid_i – *id* часового інтервалу;

ts_i – часовий проміжок.

І множина занять для розкладу, елементами якої є окремі заняття:

$$S = \{s_1, s_2, \cdots, s_n\},$$

де s_i – це заняття;

n – кількість занять.

Кожен елемент цієї множини в свою чергу складається з п'яти елементів:

– аудиторія;

– викладач;

– предмет;

– група;

– часовий проміжок.

$$s_i = \{sr_i, sp_i, sm_i, sg_i, st_i\},$$

де sr_i – аудиторія;

sp_i – викладач;

sm_i – предмет;

sg_i – група;

st_i – часовий проміжок.

3.2. Обмеження алгоритму

Проблема процесу складання розкладу полягає в тому, що розклад повинен відповідати ряду обмежень. Усі обмеження поділяють на м’які та жорсткі.

Жорсткі – це обмеження, які повинні неодмінно задовольнятися; такі які фізично не можуть бути порушені (наприклад один і той самий викладач не може бути присутній в один і той самий час у двох місцях одночасно).

М’які – це обмеження, які можна порушувати, але це порушення повинно бути зведене до мінімуму. Їхнє виконання не є таким же обов’язковим, як жорстких.

Нижче визначено жорсткі обмеження для процесу складання розкладу:

– Викладач не може бути присутнім на двох консультаціях одночасно;

– В одній аудиторії не можуть проводитися різні консультації одночасно;

– Місткість аудиторії не може бути меншою, ніж кількість присутніх у ній студентів;

– Одна група не може бути присутня на декількох консультаціях одночасно;

– Кількість консультацій на день не більше 4.

М’які обмеження, які можуть виконуватись для розкладу занять:

– У розкладі не має бути вікон;

– Викладачі можуть віддавати перевагу конкретному часу проведення консультації.

3.3. Визначення основних термінів та етапів генетичного алгоритму

Хромосома – це впорядкована послідовність генів. В даному випадку це набір консультацій у графіку. Хромосоми представляють у вигляді масиву даних, кожний елемент якого – це одна консультація в графіку.

Особина – це набір хромосом, ним буде являтись сам графік консультацій, що буде представлено у вигляді двомірного масиву хромосом.

Популяція – це кінцева множина особин, що схрещуються між собою. В даному випадку це набір сформованих графіків консультацій.

Генетичний алгоритм буде складатися з наступних етапів:

– Створення початкової популяції;
– Обчислення функції пристосованості для осіб популяції (оцінювання);
– Повторювання до виконання критерію зупинки алгоритму;
– Вибір індивідів із поточної популяції (селекція);
– Схрещення або/та мутація;
– Обчислення функції пристосовуваності для всіх осіб;
– Формування нового покоління.

3.4. Етап створення початкової популяції

Першим етапом алгоритму є створення випадковим чином деякої початкової популяції. Навіть якщо популяція виявиться абсолютно не конкурентоздатною, генетичний алгоритм все одно достатньо швидко переведе її в придатну для життя популяцію. Таким чином, на цьому кроці можна не старатися зробити надто пристосованих осіб, достатньо, щоб вони відповідали формату осіб популяції, і на них можна було порахувати функцію пристосованості. Етапи створення особини для популяції представлено на рис. 3.1.

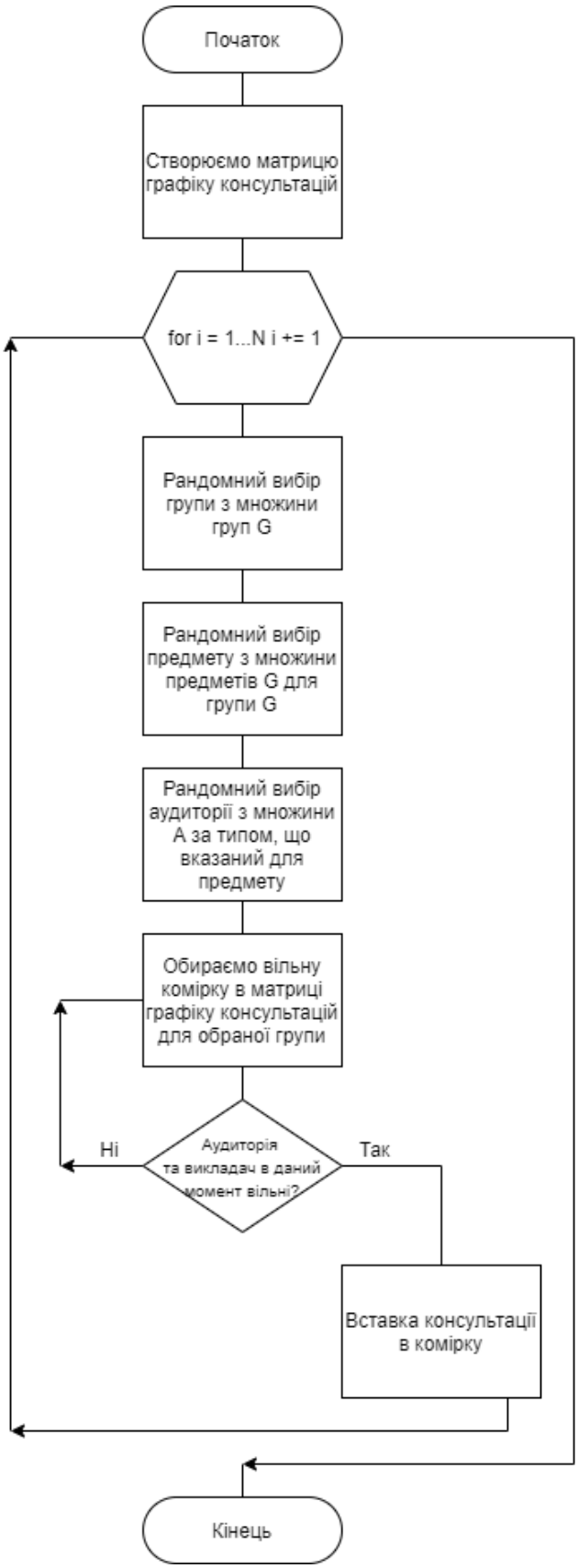


Рис. 3.1. Схема алгоритму створення початкової популяції

3.5. Схрещування

Схрещування – це один із видів оператора рекомбінації генетичного алгоритму. В науковій літературі зустрічається ще під назвою кросовер чи кросинговер. Метою схрещування є породження з наявної множини рішень нового, в якому кожна хромосома буде нащадком деяких двох елементів попередньої популяції, тобто нести в собі частково інформацію кожного батька. Допускається ситуація, коли обидва батька подані одним і тим же елементом популяції.

Схрещування відбуватиметься наступним чином:

– З батьківської популяції обираються дві особини.
– Від першого батька успадковується розташування предметів по часовим інтервалам. Від другого батька успадковується розташування аудиторій по часовим інтервалам.

Інші методи схрещування (такі як точкове схрещування) не будуть ефективними в задачі оптимізації графіку консультацій, тому що більшість створених нащадків будуть відкидатись через порушення жорстких обмежень графіку (проведення двох занять в одній аудиторії одночасно, тощо).

3.6. Мутація

Оператор мутації полягає в зміні генів у випадково вибраних позиціях. На відміну від оператора схрещування, який використовуються для поліпшення структури хромосом, метою оператора мутації є диверсифікація, тобто підвищення різноманітності пошуку і введення нових хромосом в популяцію для того, щоб більш повно досліджувати простір пошуку. Мутація ініціює різноманітність в популяції, дозволяючи проглядати більше точок в просторі пошуку і долати таким чином локальні екстремуми в ході пошуку. Необхідно відзначити, що оператор мутації є основним пошуковим оператором і існують методи, що не використовують інших операторів окрім мутації. Для вирішення даної задачі буде використовуватись проста мутація, яка використовується для бінарних, гомологічних, числових і

векторних хромосом. Буде реалізовано наступні алгоритми мутації, які будуть протестовані в ході написання коду програми, і найкращий алгоритм буде застосовано для генетичного алгоритму:

Крок 1. Зміна аудиторії:

– Копіюємо дані особини в особину-нащадка;
– Обираємо випадково консультацію;
– Змінюємо її аудиторію на 1;
– Якщо жорсткі обмеження не порушились, то завершуємо мутацію. Інакше зменшуємо значення аудиторії на 1.

Крок 2. Зміна пар:

– Копіюємо дані особини в особину-нащадка;
– Випадково обираємо дві консультації з графіку для групи;
– Міняємо їх місцями.

Крок 3. Зміна аудиторій:

– Копіюємо дані особини в особину-нащадка;
– Випадково обираємо дві консультації з графіку для групи;
– Міняємо їх аудиторії місцями.

Крок 4. Зміна часу проведення консультації:

– Копіюємо дані особини в особину-нащадка;
– Випадково обираємо консультацію з графіку для групи;
– Збільшуємо або зменшуємо випадково час його проведення.

Крок 5. Послідовний зсув консультацій:

– Копіюємо дані особини в особину-нащадка;
– Випадково обираємо день консультацій в розкладі;
– Обираємо одну чи декілька консультацій у цей день;
– Зсуваємо ці консультації на 1 пару вперед;
– Якщо консультації зсунулись на іншу консультацію, яка відбувається, то ця консультація також зсовується на 1 пару вперед. Перехід на наступний крок. Інакше завершуємо мутацію;

– Якщо ця консультація – 5 пара, то вона зсувається на першу пару наступного дня.

Крок 6. Якщо перша пара наступного дня також занята, то ця пара переноситься на місце першої пари, що була обрана для зсуву в кроці 3. Завершуємо мутацію:

– Випадкова зміна аудиторії.

На рис. 3.2 зображено схему даного алгоритму мутації.

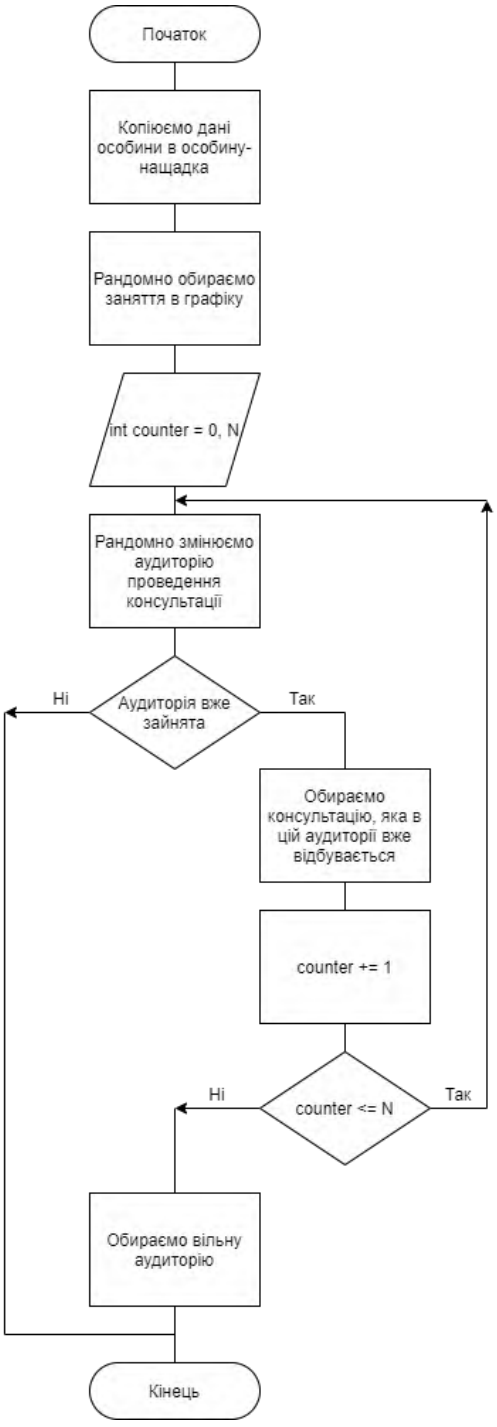


Рис. 3.2. Схема алгоритму мутації шляхом випадкової зміни аудиторії заняття

Крок 7. Зміна часу для всіх груп.

– Копіюємо дані особини в особину-нащадка;
– Випадково обираємо два стовпчики з матриці графіку консультацій (два часи проведення консультацій);
– Міняємо їх місцями.

3.7. Селекція

Селекція – це вибір тих хромосом, які будуть брати участь в створенні нащадків для наступної популяції, тобто для створення нового покоління. Такий вибір проводиться відповідно до принципу природного відбору, за яким найбільші шанси на участь в створенні нових особин мають хромосоми з найбільшими значеннями функції пристосованості. У нашому алгоритмі буде застосовано метод рулетки. Заснований на принципі колеса рулетки метод селекції вважається для генетичних алгоритмів основним методом відбору особин для батьківської популяції з метою подальшого їх перетворення генетичними операторами, такими як схрещування і мутація. Незважаючи на випадковий характер процедури селекції, батьківські особини вибираються пропорційно значенням їх функцій пристосованості: кожній хромосомі зіставлений сектор колеса рулетки, величина якого встановлюється пропорційною значенню функції пристосованості даної хромосоми:

$$l_i = \frac{f_i}{\sum_{j=1}^{N} f_j},$$

де l_i – це величина сектора рулетки (імовірність, з якою особина буде відібрана для схрещування);

i – це номер особини, що досліджується;

N – це кількість особин, що міститься в початковій популяції;

f_i – це значення цільової функції для i-ї особини.

Тому, чим більше значення функції пристосованості, тим більший сектор на колесі рулетки. А отже, тим вищий шанс, що буде обрана саме ця хромосома.

3.8. Фітнес-функція

Рішення щодо хромосоми (особини) оцінюється за допомогою значення цільової функції, тобто перевіряється, чи придатна особина для подальшого використання. Цільову функцію часто називають фітнес-функцією чи функцією придатності. Значення цієї функції оцінюється для кожної особини популяції окремо і на основі цього значення, приймається рішення використовувати цю хромосому чи перейти до етапу покращення особин популяції, за допомогою генетичних операторів схрещування та мутації. Для задачі складання і оптимізації розкладу занять було розроблено цільову функцію, що враховуватиме м'які обмеження на графік консультацій. За невиконання м'якого обмеження на розклад накладається штраф. Кожне обмеження має свій штрафний коефіцієнт, а також у кожного обмеження існує свій ступінь не виконуваності для графіку. Було обрано за ступінь не виконуваності кількість порушень і-го обмеження у графіку. Наприклад, у графіку консультацій є 6 вікон. Тоді ступінь не виконуваності для цього обмеження (у розкладі немає бути вікон) становить 6. Штраф для одного обмеження обчислюється наступним чином:

$$L_i = n_i w_i,$$

де L_i – штраф за невиконання i-го обмеження;

n_i – це ступінь не виконуваності i-го обмеження;

w_i – штрафний коефіцієнт для i-го обмеження.

Тоді формула цільової функції матиме наступний вигляд:

$$F(H_j) = \frac{1}{1 + \sum_{i=1}^{m} L_i} \xrightarrow{l} max,$$

де H_j – це досліджувана особина;

m – кількість м'яких обмежень для розкладу занять;

L_i – штраф за невиконання i-го обмеження.

Отже, чим менший штраф за невиконання м'яких обмежень, тим більшим буде значення цільової функції, а отже тим кращою буде особина. Якщо цільова функція $<< 1$, то це означає, що не виконалось якесь із м'яких обмежень, тому досліджувана особина є непридатною і підлягає покращенню своїх характеристик. Цільова функція застосовується до кожної особини популяції і спрямована на знаходження максимального значення.

3.9. Характеристика та результати роботи програмного продукту

Для формування графіку консультацій було розроблено програмний модуль, що формує графік за допомогою генетичного алгоритму. Програму реалізовано на кросплатформенній мові програмування *Python* та протестовано на вхідних даних для трьох груп кафедри комп'ютеризованих систем управління.

Вхідні дані програми можуть бути введені вручну (рис. 3.3. – 3.4.) або можна використовувати завчасно введені дані.

Набір вхідних даних складається з:

– Групи студентів. Необхідним і достатнім набором даних про групу, що потрібний для формування графіку, є назва групи та кількість студентів у ній. Наприклад, група під назвою СП-324 налічує 30 студентів;
– Аудиторій. Тут необхідно ввести номер аудиторії та кількість місць. Наприклад, аудиторія №101 може вмістити 30 студентів;
– Предметів. Наприклад, «Штучний інтелект»;
– Викладачів. Тут достатньо ввести ім'я викладача (Іванов І.І., Іванов Іван Іванович тощо);

– Часових інтервалів. Під часовим інтервалом мається на увазі час початку та закінчення заняття, а також день тижня. Наприклад, четвер, 8:00 – 9:20.

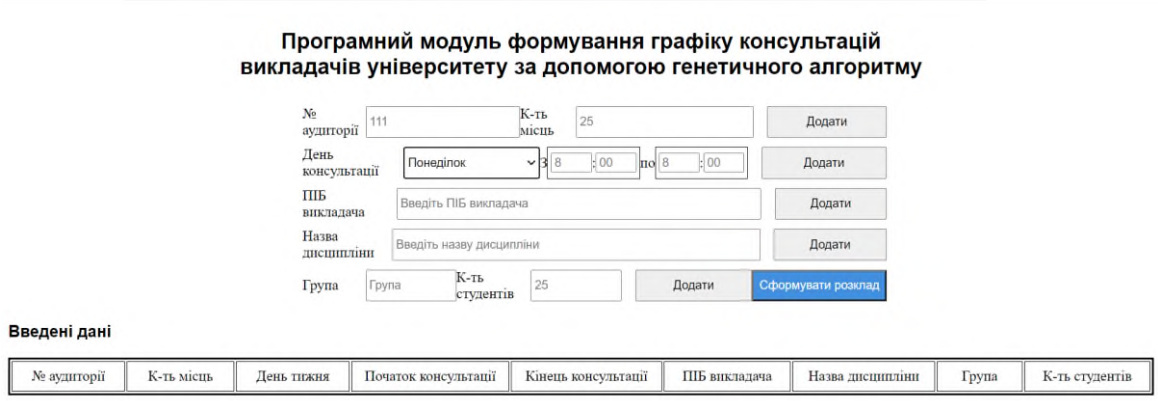
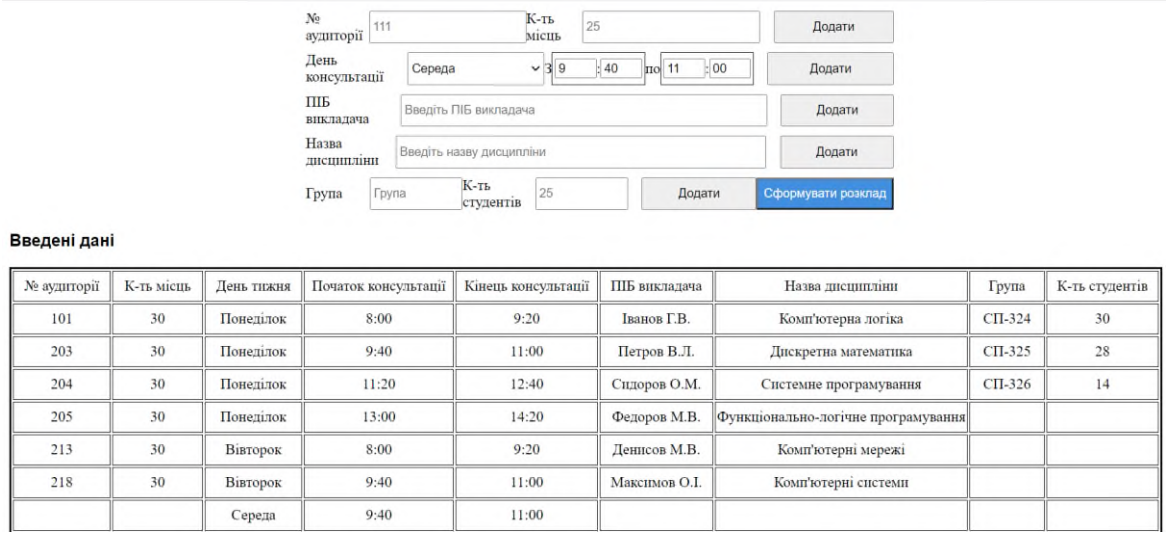


Рис. 3.3. Користувацький інтерфейс програмного модулю



№ аудиторії	К-ть місць	День тижня	Початок консультації	Кінець консультації	ПІБ викладача	Назва дисципліни	Група	К-ть студентів
101	30	Понеділок	8:00	9:20	Іванов Г.В.	Комп'ютерна логіка	СП-324	30
203	30	Понеділок	9:40	11:00	Петров В.Л.	Дискретна математика	СП-325	28
204	30	Понеділок	11:20	12:40	Сидоров О.М.	Системне програмування	СП-326	14
205	30	Понеділок	13:00	14:20	Федоров М.В.	Функціонально-логічне програмування		
213	30	Вівторок	8:00	9:20	Денисов М.В.	Комп'ютерні мережі		
218	30	Вівторок	9:40	11:00	Максимов О.І.	Комп'ютерні системи		
		Середа	9:40	11:00				

Рис. 3.4. Приклад таблиці з введеними даними

Програма виводить інформацію про значення функції пристосованості для кожної особини. На основі значень цієї функції обираються відповідні генерації. Також виводиться загальна кількість створених генерацій до того моменту, коли буде знайдене найкраще. На рис. 3.5. наведено приклад виводу таких значень.

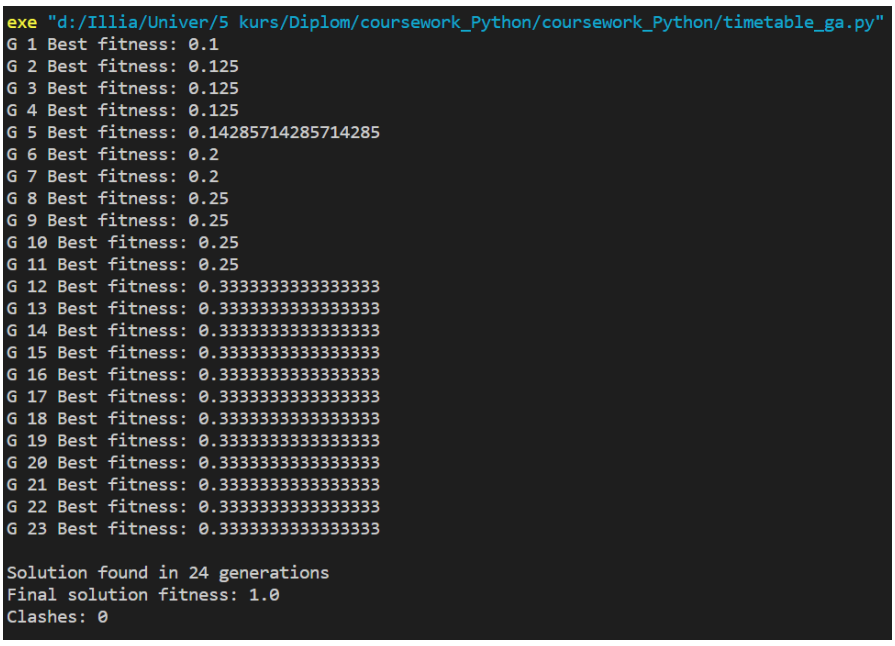


Рис. 3.5. Значення фітнес-функції для кожного покоління

Програма зберігає згенерований розклад консультацій у файлі *.xlsx* у вигляді таблиці, що містить наступний набір даних:

– Назва предмета;

– Номер аудиторії;

– Ім’я викладача;

– Час початку та закінчення консультації.

Результати представлені на рисунках 3.6 – 3.8.

	A	B	C	D	E	F
1	Понеділок	Вівторок	Середа	Четвер	П'ятниця	СП-324
2	Системне програмування 219 Петров В.Л. 8:00 - 9:20	Комп'ютерні системи 219 Сидоров О.М. 8:00 - 9:20	Комп'ютерна логіка 311 Іванов Г.В. 8:00 - 9:20	Штучний інтелект 304 Максимов О.І. 8:00 - 9:20		
3	Системне програмне забезпечення 306 Федоров М.В. 9:40 - 11:00				Дискретна математика 306 Іванов Г.В. 9:40 - 11:00	
4				Об'єктно-орієнтоване програмування 305 Федоров М.В. 11:20 - 12:40		
5	Функціонально-логічне програмування 213 Денисов М.В. 13:00 - 14:20	Бази даних 311 Максимов О.І. 13:00 - 14:20			Комп'ютерні мережі 101 Сидоров О.М. 13:00 - 14:20	

Рис. 3.6. Розклад консультацій для групи СП-324

	A	B	C	D	E	F
6	Понеділок	Вівторок	Середа	Четвер	П'ятниця	СП-325
7				Комп'ютерні системи 213 Сидоров О.М. 8:00 - 9:20	Об'єктно-орієнтоване програмування 304 Федоров М.В. 8:00 - 9:20	
8		Системне програмне забезпечення 205 Федоров М.В. 9:40 - 11:00	Бази даних 311 Максимов О.І. 9:40 - 11:00		Системне програмування 205 Петров В.Л. 9:40 - 11:00	
9	Комп'ютерна логіка 311 Іванов Г.В. 11:20 - 12:40				Штучний інтелект 218 Максимов О.І. 11:20 - 12:40	
10		Комп'ютерні мережі 218 Сидоров О.М. 13:00 - 14:20		Функціонально-логічне програмування 203 Денисов М.В. 13:00 - 14:20	Дискретна математика 305 Іванов Г.В. 13:00 - 14:20	

Рис. 3.7. Розклад консультацій для групи СП-325

	A	B	C	D	E	F
11	Понеділок	Вівторок	Середа	Четвер	П'ятниця	СП-326
12	Комп'ютерні мережі 101 Сидоров О.М. 8:00 - 9:20					
13	Комп'ютерна логіка 311 Іванов Г.В. 9:40 - 11:00		Дискретна математика 204 Іванов Г.В. 9:40 - 11:00	Об'єктно-орієнтоване програмування 213 Федоров М.В. 9:40 - 11:00	Системне програмне забезпечення 219 Федоров М.В. 9:40 - 11:00	
14	Штучний інтелект 316 Максимов О.І. 11:20 - 12:40				Функціонально-логічне програмування 219 Денисов М.В. 11:20 - 12:40	
15			Комп'ютерні системи 313 Сидоров О.М. 13:00 - 14:20	Системне програмування 311 Петров В.Л. 13:00 - 14:20	Бази даних 303 Максимов О.І. 13:00 - 14:20	

Рис. 3.8. Розклад консультацій для групи СП-326

3.10. Висновки до розділу

У даному розділі було детально описано об’єкт оптимізації генетичного алгоритму – графік консультацій. Було визначено з яких множин він буде складатись та як ці множини буде поєднано для його утворення. Також було визначено обмеження, які потрібно враховувати при складанні графіку, а саме ті, дотримання яких є обов’язковим для успішного складання графіку, і ті, виконання яких є бажаним для того, щоб графік консультацій був близьким до оптимального.

Також був розроблений генетичний алгоритм оптимізації графіку консультацій. Було описано задачу оптимізації у термінах генетичного алгоритму, визначено основні етапи алгоритму (формування початкової популяції, селекція, схрещування, мутація, вибір найкращого варіанту) та описано дії, що виконуватимуться на кожному з цих етапів.

Встановлено набір обмежень, що висуваються до готового графіку. Проблема процесу складання розкладу полягає в тому, що розклад повинен відповідати ряду обмежень. Усі обмеження поділяють на м’які та жорсткі.

Жорсткі – це обмеження, які повинні неодмінно задовольнятися; такі які фізично не можуть бути порушені (наприклад один і той самий викладач не може бути присутній в один і той самий час у двох місцях одночасно).

М’які – це обмеження, які можна порушувати, але це порушення повинно бути зведене до мінімуму. Їхнє виконання не є таким же обов’язковим, як жорстких.

Перелік жорстких обмежень, що накладаються на графік, виглядає наступним чином:

– Викладач не може бути присутнім на двох консультаціях одночасно;
– В одній аудиторії не можуть проводитися різні консультації одночасно;
– Місткість аудиторії не може бути меншою, ніж кількість присутніх у ній студентів;
– Одна група не може бути присутня на декількох консультаціях одночасно;
– Кількість консультацій на день не більше 4.

М’які обмеження, які можуть виконуватись для розкладу занять:

– У розкладі не має бути вікон;
– Викладачі можуть віддавати перевагу конкретному часу проведення консультації.

З’ясовано набір та описано набір вхідних даних, необхідних для формування розкладу, а саме:

– Групи студентів;
– Аудиторії;
– Предмети;
– Викладачі;
– Часові інтервали (заняття).

Було розроблено програмний продукт, що на основі даних, введених диспетчером, формує графік консультацій викладачів за допомогою генетичного алгоритму та зберігає його у файлі, який містить графік для кожної з груп.

ВИСНОВКИ

Метою роботи було розроблення програмного модулю формування графіку консультацій викладачів університету за допомогою генетичного алгоритму.

Вже довгий час питання про розробку та автоматизацію якісного алгоритму складання розкладу ВНЗ залишається одним із найважливіших в сфері освіти. Добре складений графік впливає як на засвоюваність матеріалів студентами, так і на якість навчання викладачами. Також система повинна бути економічно вигідною для свого закладу. Процес складання розкладу – це надзвичайно тривалий та непростий процес, який потребує від людей, які його складають, певних вмінь та знань. Його математично можна формалізувати як складну оптимізаційну задачу, для вирішення якої сьогодні існує ряд методів.

Для досягнення результату був проведений аналіз актуальності даного питання та історії теорії розкладу. Було розглянуто основні поняття еволюційних методів для розв’язання оптимізаційних задач, детально описано один з них, а саме генетичний алгоритм, і можна сказати, що цей алгоритм заснований на аналогії з природними еволюційними процесами. Його перевагою є те, що не висуваються додаткові вимоги до виду цільової функції, алгоритм на кожній ітерації працює з множиною рішень, що дозволяє в багатьох випадках більш детально в порівнянні з градієнтними методами багатовимірної нелінійної безумовної оптимізації аналізувати простір пошуку. Загалом, генетичний алгоритм являється потужним інструментом для вирішення складних оптимізаційних задач і його практичне застосування відбувається сьогодні все частіше і частіше.

На основі отриманих даних після генерації розкладу, а також при порівнянні з ручним методом, можна зробити висновок про досягнення поставленої мети: час, витрачений на створення графіку системою значно менше часу, що витрачається на ручне складання розкладу незалежно від способу ручної генерації.

Дослідивши, як застосовується генетичний алгоритм для проблеми складання графіку в університетах, можна сказати, що цей метод має ряд переваг у порівнянні із іншими:

– відсутність необхідності у специфічних знаннях про вирішувану задачу;
– концептуальна простота та прозорість реалізації;
– можливість розпаралелювання;
– простота кодування вхідної і вихідної інформації;
– можливість застосування до великого кола задач без внесення серйозних змін у внутрішню структуру методу.

В ході виконання роботи було досліджено застосування генетичного алгоритму для задачі створення і оптимізації графіку консультацій викладачів в університеті.

На даному етапі було виконано наступні завдання:

– Зроблено огляд предметної області;
– Складено загальний опис генетичного алгоритму;
– Складено загальний опис досліджуваного об’єкту (графіку консультацій);
– Сформульовані задача оптимізації графіку консультацій в термінах генетичного алгоритму та критерій якості графіку (цільова функція, що залежить обернено пропорційно від об’єму порушень м’яких обмежень, що накладаються на графік, значення якої потрібно максимізувати для отримання оптимального розкладу);
– Сформовано алгоритм створення початкової популяції;
– Визначено методи селекції для відбору батьківської популяції (турнірний відбір, метод рулетки та метод рангів), серед яких у процесі написання коду шляхом тестування буде вибрано найкращий метод;
– Визначено спосіб схрещування батьківських особин;
– Розглянуто різні способи мутації особин (зміна значень аудиторій, зміна пар, зміна аудиторій місцями, зміна часу, зсув занять, випадкова зміна аудиторії, зміна часових інтервалів для усіх груп одночасно), найкращий з

яких буде обрано в ході проектування архітектури програми шляхом тестування;

– На основі сформульованого генетичного алгоритму було реалізовано програмний продукт, що формує графік консультацій за допомогою даного алгоритму та зберігає його у вигляді електронної таблиці. Програму реалізовано на мові програмування *Python*;
– Розроблене рішення було досліджено і протестовано на різних об’ємах даних. Було зроблено висновок, що розроблений алгоритм виконує поставлену задачу достатньо швидко, а отже, може бути застосований для формування графіку консультацій в реальних умовах.

Результати роботи алгоритму зберігаються у формі, придатній для подальшого зручного використання та оброблення. Закладена можливість подальшого удосконалення алгоритму та розширення його функціональності.

Подальшим розвитком програмного модулю може бути можливість створення веб-версії для зручної роботи, зберігання необхідної інформації у базі даних та надання користувачам можливості особисто вказувати критерії складання розкладу.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. *Gantt H.L. ASME Transactions*, 1903, 24, P. 1322–1336.
2. Матковский И. Генри Гант // Знаменитости. – [Електронний ресурс]. – Режим доступу: *http://www.peoples.ru/technics/engineer/henry_gantt/*.
3. *Bellman R. Mathematical aspects of scheduling theory // Journal of the Society of Industrial and Applaid Mathematics*. 1956. *Vol. 4. P.* 168–205.
4. Беллман Р. Динамическое программирование М.: ИЛ, 1960. 400 с.
5. Задача теории расписания – [Електронний ресурс]. – Режим доступу: *https://sites.google.com/site/askerpro/4*
6. Кононов А.В. Актуальные задачи теории расписаний: вычислительная сложность и приближенные алгоритмы. – М.: Институт математики им. С.Л. Соболева; Новосибирск, 2014. - 196 с.
7. *Conway R.W., Maxwell W.L., Miller L.W. Theory of Scheduling. Addison-Wesley, Reading, MA*. 1967.
8. Танаев В.С., Шкурба В.В. Введение в теорию расписаний. М.: Наука, 1975.
9. *Graham R.L., Lawler E.L., Lenstra J.K., Rinnooy Kan A.H.G. Optimization and approximation in determenistic scheduling: a servey // Annals of Discrete Mathematics* — 1979. — *Vol. 5 — P.* 287–326.
10. *Edmonds J. Paths, trees, and flowers // Canadian Journal of Mathematics.* — 1965. — *Vol. 7 — P.* 449–467.
11. *Cook S. A. The complexity of theorem proving procedures // Proceedings of the 3rd Annual ACM Symposium on the theory of Computing* — 1971. — P. 151–158.
12. *Garey M. and Johnson D. Computers and Intractability: A Guide to the theory of NP-completeness. — San Francisco, CA: W.H. Freemann and Company, 1979.*
13. *Papadimitriou C., Yannakakis M. Optimization, approximation and complexity classes // Journal of Computer and System Science.* — 1991. — *Vol 43. — P.* 425–440.

14. Расписание занятий как организационный документ, определяющий режим работы ОУ. – [Електронний ресурс]. – Режим доступу: *http://festival.1september.ru/articles/519295/*
15. Лазарев А.А., Гафаров Е.Р. Теория расписаний. Задачи и алгоритмы. М.: Московский государственный университет им. М.В. Ломоносова; Москва, 2011, 222 с.
16. Бойко О.М. Еволюційна технологія розв'язування задачі складання розкладів навчальних занять/ Бойко О.М. // Штучний інтелект. – 2006. – №.3. – С. 341-348.
17. Композиционный генетический алгоритм для составления расписания занятий. – [Електронний ресурс]. – Режим доступу: *http://cyberleninka.ru/article/n/kompozitsionnyy-geneticheskiy-algoritm-sostavleniya-raspisaniya-uchebnyh-zanyati*.
18. Проблемы составления расписания в ВУЗе. – [Електронний ресурс]. – Режим доступу: *http://www.pulsar.ru/pressa/2004/* .
19. Бевз С. В. Розробка автоматизованої системи формування розкладу магістратури / Бевз С. В., Войтко В. В., Бурбело С. М., Шоботенко А. М. // Наукові праці ВНТУ. – 2009. – №1. – С. 1-10.
20. Береговых Ю.В. Алгоритм составления расписания занятий / Ю.В. Береговых, Н.А. Володин // Искусственный интеллект. 2009. № 2. С. 50-57.
21. ГОСТ 19.701–90 ЕСПД. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения.
22. ДСТУ ГОСТ 7.1:2006. Бібліографічний запис. Бібліографічний опис. Загальні вимоги та правила складання.
23. ГОСТ 2.106–96 ЕСКД "Текстовые документы".
24. ДСТУ 3008–95 "Документація. Звіти у сфері науки і техніки. Структура і правила оформлення".
25. Бойченко С.В., Іванченко О.В. Положення про дипломні роботи (проекти) випускників Національного авіаційного університету. – Київ: НАУ, 2017. – 63 с.

Додаток А

Лістинг коду програмного модулю

Файл *timetable_ga.py*:

```
from genetic_algorithm import GeneticAlgorithm
from population import Population
from timetable import Timetable
import xlsxwriter

def initialize_timetable():
    timetable = Timetable()
    timetable.add_room(1, "101", 30)
    timetable.add_room(2, "203", 30)
    timetable.add_room(3, "204", 30)
    timetable.add_room(4, "205", 30)
    timetable.add_room(5, "213", 30)
    timetable.add_room(6, "218", 30)
    timetable.add_room(7, "219", 30)
    timetable.add_room(8, "303", 30)
    timetable.add_room(9, "304", 70)
    timetable.add_room(10, "305", 30)
    timetable.add_room(11, "306", 30)
    timetable.add_room(12, "311", 70)
    timetable.add_room(13, "313", 70)
    timetable.add_room(14, "316", 70)
    timetable.add_timeslot(1, "Понеділок, 8:00 - 9:20")
    timetable.add_timeslot(2, "Понеділок, 9:40 - 11:00")
    timetable.add_timeslot(3, "Понеділок, 11:20 - 12:40")
    timetable.add_timeslot(4, "Понеділок, 13:00 - 14:20")
    timetable.add_timeslot(5, "Вівторок, 8:00 - 9:20")
```

timetable.add_timeslot(6, "Вівторок, 9:40 - 11:00")
timetable.add_timeslot(7, "Вівторок, 11:20 - 12:40")
timetable.add_timeslot(8, "Вівторок, 13:00 - 14:20")
timetable.add_timeslot(9, "Середа, 8:00 - 9:20")
timetable.add_timeslot(10, "Середа, 9:40 - 11:00")
timetable.add_timeslot(11, "Середа, 11:20 - 12:40")
timetable.add_timeslot(12, "Середа, 13:00 - 14:20")
timetable.add_timeslot(13, "Четвер, 8:00 - 9:20")
timetable.add_timeslot(14, "Четвер, 9:40 - 11:00")
timetable.add_timeslot(15, "Четвер, 11:20 - 12:40")
timetable.add_timeslot(16, "Четвер, 13:00 - 14:20")
timetable.add_timeslot(17, "П'ятниця, 8:00 - 9:20")
timetable.add_timeslot(18, "П'ятниця, 9:40 - 11:00")
timetable.add_timeslot(19, "П'ятниця, 11:20 - 12:40")
timetable.add_timeslot(20,"П'ятниця,13:00 - 14:20")

timetable.add_professor(1, "Іванов Г.В.")
timetable.add_professor(2, "Петров В.Л.")
timetable.add_professor(3, "Сидоров О.М.")
timetable.add_professor(4, "Федоров М.В.")
timetable.add_professor(5, "Денисов М.В.")
timetable.add_professor(6, "Максимов О.І.")

timetable.add_module(1, "pp1", "Комп'ютерна логіка", [1])
timetable.add_module(2, "pp2", "Дискретна математика", [1, 8])
timetable.add_module(3, "eit1", "Системне програмування", [2])
timetable.add_module(4, "eit2", "Функціонально-логічне програмування", [5])
timetable.add_module(5, "qa1", "Комп'ютерні мережі", [3])
timetable.add_module(6, "qa2", "Комп'ютерні системи", [3])
timetable.add_module(7, "web1", "Системне програмне забезпечення", [4])

```
        timetable.add_module(8, "web2", "Об'єктно-орієнтоване програмування", [4, 7])
        timetable.add_module(9, "spas1", "Штучний інтелект", [6])
        timetable.add_module(10, "spas2", "Бази даних", [6])
        timetable.add_group(1, "СП-124", 30, [1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
        timetable.add_group(2, "СП-224", 28, [1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
        timetable.add_group(3, "СП-324", 14, [1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
        return timetable

    def main():
        timetable = initialize_timetable()
        ga = GeneticAlgorithm(100, 0.01, 0.9, 2, 5)
        population = ga.init_population(timetable)
        ga.eval_population(population, timetable)
        generation = 1
        while ga.is_termination_condition_met(generations_count=generation, max_generations=1000) is False and ga.is_termination_condition_met(population=population) is False:
            print('G {} Best fitness: {}'.format(generation, population.get_fittest(0).fitness))
            population = ga.crossover_population(population)
            population = ga.mutate_population(population, timetable)
            ga.eval_population(population, timetable)
            generation += 1
        timetable.create_classes(population.get_fittest(0))
        print()
        print('Solution found in {} generations'.format(generation))
        print('Final solution fitness: {}'.format(population.get_fittest(0).fitness))
        print('Clashes: {}'.format(timetable.calc_clashes()))
        # Print classrooms
        print()
```

```
classrooms = timetable.classes
# class_index = 1
# for best_classroom in classrooms:
#     print('Classroom {}:'.format(class_index))
#     print(timetable.get_module(best_classroom.module_id).module_name)
#     print(timetable.get_group(best_classroom.group_id).group_name)
#     print(timetable.get_room(best_classroom.room_id).room_number)
#     print(timetable.get_professor(best_classroom.professor_id).professor_name)
#     print(timetable.get_timeslot(best_classroom.timeslot_id).timeslot)
#     print('-----')
#     class_index += 1

workbook = xlsxwriter.Workbook('TIMETABLE.xlsx')
worksheet = workbook.add_worksheet()
bold_center = workbook.add_format({'bold': True, 'align': 'center'})
bold_red = workbook.add_format({'bold': True, 'color': 'red'})
center = workbook.add_format({'align': 'center'})
text_wrap = workbook.add_format({'text_wrap': True, 'valign': 'top'})

i = 0
group_index = 0
timeslot_index = 0
while (i < 5 * len(timetable.groups)):
    if (i % 5 == 0):
        worksheet.write_string(i, 0, 'Понеділок', bold_center)
        worksheet.write_string(i, 1, 'Вівторок', bold_center)
        worksheet.write_string(i, 2, 'Середа', bold_center)
        worksheet.write_string(i, 3, 'Четвер', bold_center)
        worksheet.write_string(i, 4, "П'ятниця", bold_center)
```

```
                worksheet.write_string(i, 5, timetable.get_group(group_index + 1).group_name, bold_red)
                i += 1
                group_index += 1
            timeslot_index = i % 5
            for j in range(5):
                worksheet.write_string(i, j, find_classroom(group_index, timeslot_index, timetable, classrooms), center)
                timeslot_index += 4
            i += 1
        workbook.close()

    def find_classroom(group_index, timeslot_index, timetable, classrooms):
        for best_classroom in classrooms:
            if (best_classroom.group_id == group_index and best_classroom.timeslot_id == timeslot_index):
                return str(timetable.get_module(best_classroom.module_id).module_name) + '\n' + str(timetable.get_room(best_classroom.room_id).room_number) + '\n' + str(timetable.get_professor(best_classroom.professor_id).professor_name) + '\n' + str(timetable.get_timeslot(best_classroom.timeslot_id).timeslot.split(',')[1])
        return ''

    if __name__ == "__main__":
        main()
```

Файл *timetable.py*:

```
    from classroom import Classroom
    from room import Room
    from professor import Professor
    from module import Module
```

```
from group import Group
from timeslot import Timeslot
from random import random
```

Файл *timetable.py*:

```
class Timetable:
    _classes = []
    _num_classes = 0

    def __init__(self, cloneable=None):
        if cloneable is None:
            self._rooms = {}
            self._professors = {}
            self._modules = {}
            self._groups = {}
            self._timeslots = {}
        elif cloneable is not None:
            self._rooms = cloneable.rooms
            self._professors = cloneable.professors
            self._modules = cloneable.modules
            self._groups = cloneable.groups
            self._timeslots = cloneable.timeslots

    @property
    def groups(self):
        return self._groups

    @property
    def timeslots(self):
        return self._timeslots
```

```
@property
def modules(self):
    return self._modules

@property
def professors(self):
    return self._professors

@property
def rooms(self):
    return self._rooms

def add_room(self, room_id, room_name, capacity):
    self._rooms[room_id] = Room(room_id, room_name, capacity)

def add_professor(self, professor_id, professor_name):
    self._professors[professor_id] = Professor(
        professor_id, professor_name)

def add_module(self, module_id, module_code, module_name, professor_ids):
    self._modules[module_id] = Module(
        module_id, module_code, module_name, professor_ids)

def add_group(self, group_id, group_name, group_size, module_ids):
    self._groups[group_id] = Group(
        group_id, group_name, group_size, module_ids)
    self._num_classes = 0

def add_timeslot(self, timeslot_id, timeslot):
```

```
    self._timeslots[timeslot_id] = Timeslot(timeslot_id, timeslot)

def get_room(self, room_id):
    if room_id not in self._rooms:
        print("Rooms doesn't contain key " + room_id)
    return self._rooms[room_id]

def get_random_room(self):
    rooms_array = list(self._rooms.values())
    room = rooms_array[(int)(len(rooms_array) * random())]
    return room

def get_professor(self, professor_id):
    return self._professors[professor_id]

def get_module(self, module_id):
    return self._modules[module_id]

def get_group_modules(self, group_id):
    group = self._groups[group_id]
    return group.module_ids

def get_group(self, group_id):
    return self._groups[group_id]

def get_groups_as_array(self):
    return list(self._groups.values())

def get_timeslot(self, timeslot_id):
    return self._timeslots[timeslot_id]
```

```
def get_random_timeslot(self):
    timeslot_array = list(self._timeslots.values())
    timeslot = timeslot_array[(int)(len(timeslot_array) * random())]
    return timeslot

@property
def classes(self):
    return self._classes

def get_num_classes(self):
    if self._num_classes > 0:
        return self._num_classes
    num_classes = 0
    groups = list(self._groups.values())
    for group in groups:
        num_classes += len(group.module_ids)
    self._num_classes = num_classes
    return self._num_classes

def create_classes(self, individual):
    classes = [None] * self.get_num_classes()
    chromosome = individual.chromosome
    chromosome_pos = 0
    class_index = 0
    for group in self.get_groups_as_array():
        module_ids = group.module_ids
        for module_id in module_ids:
            classes[class_index] = Classroom(
                class_index, group.group_id, module_id)
```

```
                # Add timeslot
                classes[class_index].timeslot_id = chromosome[chromosome_pos]
                chromosome_pos += 1
                # Add room
                classes[class_index].room_id = chromosome[chromosome_pos]
                chromosome_pos += 1
                # Add professor
                classes[class_index].professor_id = chromosome[chromosome_pos]
                chromosome_pos += 1
                class_index += 1
            self._classes = classes

        def calc_clashes(self):
            clashes = 0
            for class_a in self._classes:
                # Check room capacity
                room_capacity = self.get_room(class_a.room_id).room_capacity
                group_size = self.get_group(class_a.group_id).group_size
                if room_capacity < group_size:
                    clashes += 1
                # Check if room is taken
                for class_b in self._classes:
                    if class_a.room_id == class_b.room_id and class_a.timeslot_id ==
class_b.timeslot_id and class_a.class_id != class_b.class_id:
                        clashes += 1
                        break
                    if class_a.room_id != class_b.room_id and class_a.timeslot_id ==
class_b.timeslot_id and class_a.group_id == class_b.group_id and class_a.class_id !=
class_b.class_id:
                        clashes += 1
```

```
                    break
                if class_a.professor_id == class_b.professor_id and class_a.timeslot_id
== class_b.timeslot_id and class_a.class_id != class_b.class_id:
                    clashes += 1
                    break

        return clashes
```

Файл *genetic_algorithm.py*:

```
from population import Population
from timetable import Timetable
from individual import Individual
from random import random


class GeneticAlgorithm:
    _tournament_size = 0

    def __init__(self, population_size, mutation_rate, crossover_rate, elitism_count,
tournament_size):
        self._population_size = population_size
        self._mutation_rate = mutation_rate
        self._crossover_rate = crossover_rate
        self._elitism_count = elitism_count
        GeneticAlgorithm._tournament_size = tournament_size

    def init_population(self, timetable):
        # Initialize population
        population = Population(self._population_size, timetable)
        return population
```

```
def is_termination_condition_met(self, population=None,
                                 generations_count=None,
                                 max_generations=None):
    if population is not None:
        return population.get_fittest(0).fitness == 1.0
    elif generations_count is not None and max_generations is not None:
        return generations_count > max_generations

def calc_fitness(self, individual, timetable):
    thread_timetable = Timetable(timetable)
    thread_timetable.create_classes(individual)
    clashes = thread_timetable.calc_clashes()
    fitness = 1 / float((clashes + 1))
    individual.fitness = fitness
    return fitness

def eval_population(self, population, timetable):
    population_fitness = 0
    # Loop over population evaluating individuals and summing population fitness
    for individual in population.individuals:
        population_fitness += self.calc_fitness(individual, timetable)
    population.population_fitness = population_fitness

@staticmethod
def select_parent(population):
    tournament = Population(GeneticAlgorithm._tournament_size)
    population.shuffle()
    for i in range(GeneticAlgorithm._tournament_size):
        tournament_individual = population.get_individual(i)
```

```
            tournament.set_individual(i, tournament_individual)
        return tournament.get_fittest(0)

    def mutate_population(self, population, timetable):
        new_population = Population(self._population_size)
        for population_index in range(population.size()):
            individual = population.get_fittest(population_index)
            random_individual = Individual(timetable=timetable)
            for gene_index in range(individual.chromosome_length):
                if population_index > self._elitism_count:
                    if self._mutation_rate > random():
                        individual.set_gene(gene_index,
random_individual.get_gene(gene_index))
            new_population.set_individual(population_index, individual)
        return new_population

    def crossover_population(self, population):
        new_population = Population(population.size())
        for population_index in range(population.size()):
            parent1 = population.get_fittest(population_index)
            if self._crossover_rate > random() and population_index >=
self._elitism_count:
                offspring = Individual(chromosome_length=parent1.chromosome_length)
                parent2 = GeneticAlgorithm.select_parent(population)
                for gene_index in range(parent1.chromosome_length):
                    if 0.5 > random():
                        offspring.set_gene(gene_index, parent1.get_gene(gene_index))
                    else:
                        offspring.set_gene(gene_index, parent2.get_gene(gene_index))
                new_population.set_individual(population_index, offspring)
```

```
            else:
                new_population.set_individual(population_index, parent1)
        return new_population
```

Файл *individual.py*:

```
class Individual:

    def __init__(self, timetable=None, chromosome_length=None):
        if timetable is not None:
            num_classes = timetable.get_num_classes()
            _chromosome_length = num_classes * 3
            new_chromosome = [None] * _chromosome_length
            chromosome_index = 0
            for group in timetable.get_groups_as_array():
                for module_id in group.module_ids:
                    timeslot_id = timetable.get_random_timeslot().timeslot_id
                    new_chromosome[chromosome_index] = timeslot_id
                    chromosome_index += 1
                    room_id = timetable.get_random_room().room_id
                    new_chromosome[chromosome_index] = room_id
                    chromosome_index += 1
                    module = timetable.get_module(module_id)
                    new_chromosome[chromosome_index] =
module.get_random_professor_id()
                    chromosome_index += 1
            self._chromosome = new_chromosome
            self._fitness = -1
        elif chromosome_length is not None:
            individual = [None] * chromosome_length
            for gene in range(chromosome_length):
```

```
            individual[gene] = gene
        self._chromosome = individual
        self._fitness = -1

@property
def chromosome(self):
    return self._chromosome

@property
def chromosome_length(self):
    return len(self._chromosome)

def set_gene(self, offset, gene):
    self._chromosome[offset] = gene

def get_gene(self, offset):
    return self._chromosome[offset]

@property
def fitness(self):
    return self._fitness

@fitness.setter
def fitness(self, value):
    self._fitness = value

def __str__(self):
    output = ''
    for gene in range(len(self._chromosome)):
        output += self._chromosome[gene] + ','
```

```
        return output

    def contains_gene(self, gene):
        for i in range(len(self._chromosome)):
            if self._chromosome[i] == gene:
                return True
        return False
```

Файл *population.py*

```
from individual import Individual
from random import randint

class Population:
    _population_fitness = -1

    def __init__(self, population_size, timetable=None):
        self._population = [None] * population_size
        if timetable is not None:
            for individual_count in range(population_size):
                individual = Individual(timetable=timetable)
                self._population[individual_count] = individual

    @property
    def individuals(self):
        return self._population

    def get_fittest(self, offset):
        self._population.sort(key=lambda x: x.fitness, reverse=True)
        return self._population[offset]
```

```
    @property
    def population_fitness(self):
        return self._population_fitness

    @population_fitness.setter
    def population_fitness(self, value):
        self._population_fitness = value

    def size(self):
        return len(self._population)

    def set_individual(self, offset, individual):
        self._population[offset] = individual
        return self._population[offset]

    def get_individual(self, offset):
        return self._population[offset]

    def shuffle(self):
        for i in range(len(self._population)-1, 0, -1):
            index = randint(0, i)
            a = self._population[index]
            self._population[index] = self._population[i]
            self._population[i] = a
```

Файл *timeslot.py:*

```
class Timeslot:
    def __init__(self, timeslot_id, timeslot):
        self._timeslot_id = timeslot_id
        self._timeslot = timeslot
```

```
    @property
    def timeslot_id(self):
        return self._timeslot_id

    @property
    def timeslot(self):
        return self._timeslot
```

Файл *room.py:*

```
class Room:
    def __init__(self, room_id, room_number, room_capacity):
        self._room_id = room_id
        self._room_number = room_number
        self._room_capacity = room_capacity

    @property
    def room_id(self):
        return self._room_id

    @property
    def room_number(self):
        return self._room_number

    # getRoomCapacity Java
    @property
    def room_capacity(self):
        return self._room_capacity
```

Файл *proffesor.py:*

```
class Professor:
    def __init__(self, professor_id, professor_name):
        self._professor_id = professor_id
        self._professor_name = professor_name

    @property
    def professor_id(self):
        return self._professor_id

    @property
    def professor_name(self):
        return self._professor_name
```

Файл *module.py:*

```
from random import random

class Module:
    def __init__(self, module_id, module_code, module_name, professor_ids):
        self._module_id = module_id
        self._module_code = module_code
        self._module_name = module_name
        self.professor_ids = professor_ids

    @property
    def module_id(self):
        return self._module_id

    @property
    def module_code(self):
```

```
        return self._module_code

    # getModuleName in Java
    @property
    def module_name(self):
        return self._module_name

    def get_random_professor_id(self):
        random_id = (int)(len(self.professor_ids) * random())
        professor_id = self.professor_ids[random_id]
        return professor_id
```

Файл *group.py:*

```
class Group:
    def __init__(self, group_id, group_name, group_size, module_ids):
        self._group_id = group_id
        self._group_name = group_name
        self._group_size = group_size
        self._module_ids = module_ids

    @property
    def group_id(self):
        return self._group_id

    @property
    def group_name(self):
        return self._group_name

    @property
    def group_size(self):
```

```
        return self._group_size

    @property
    def module_ids(self):
        return self._module_ids
```

Файл *classroom.py:*

```
class Classroom:
    def __init__(self, class_id, group_id, module_id):
        self._class_id = class_id
        self._group_id = group_id
        self._module_id = module_id
    @property
    def class_id(self):
        return self._class_id
    @property
    def group_id(self):
        return self._group_id
    @property
    def module_id(self):
        return self._module_id
    @property
    def professor_id(self):
        return self._professor_id
    @property
    def timeslot_id(self):
        return self._timeslot_id
    @property
    def room_id(self):
        return self._room_id
```

```
@professor_id.setter
def professor_id(self, value):
    self._professor_id = value

@timeslot_id.setter
def timeslot_id(self, value):
    self._timeslot_id = value

@room_id.setter
def room_id(self, value):
    self._room_id = value
```